

国际信息工程先进技术译丛

WILEY

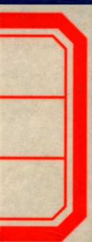
信号处理系统的FPGA实现

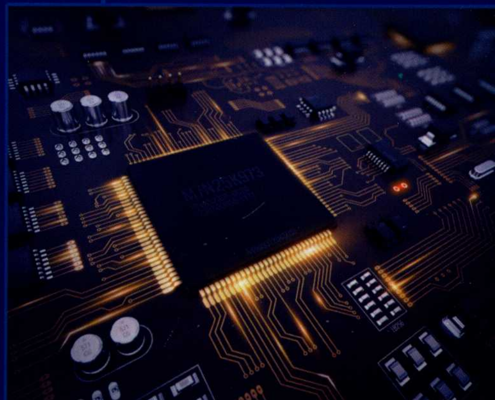
FPGA-based Implementation of Signal Processing Systems

[英] 罗杰·伍兹（Roger Woods）
约翰·麦考利斯特（John McAllister）
盖伊·莱特博迪（Gaye Lightbody）
易英（Ying Yi） 著

李争平 曾勇波 黄明 白文乐 译







机械工业出版社
CHINA MACHINE PRESS

J技术译丛

信号处理系统的FPGA实现

[英] 罗杰·伍兹(Roger Woods)
约翰·麦考利斯特(John McAllister)
盖伊·莱特博迪(Gaye Lightbody)
易英(Ying Yi) 著

李争平 曾勇波 黄 明 白文乐 译



机械工业出版社

本书从数字信号处理技术、定点、浮点的运算到微处理器技术、FPGA技术的演进过程出发，以结构原理为基础，讨论了各种典型FPGA系列的特点，为器件选型提供指导。同时，详细讲解了FPGA的DSP快速设计流程、先进的综合工具、性能测试分析工具、性能优化技术、低功耗设计及可重配置技术。本书还提供了诸如自适应波束形成器等大量翔实的FPGA实现DSP的复杂实例，为开发者提供便利。全书共14章，涵盖了DSP基础知识、DSP处理器的发展、各系列FPGA介绍、FPGA实现DSP的方法、IRIS综合技术、IP核技术、异构FPGA模型化设计、自适应波束形成器、低功耗技术及可重配置技术。全面展现了FPGA的特点及各种主流开发技术。

本书适合有一定电路与信息处理基础的研究生或工程开发人员使用。

图书在版编目（CIP）数据

信号处理系统的FPGA实现/（英）罗杰·伍兹（Roger Woods）等著；李争平等译. —北京：机械工业出版社，2017.2

（国际信息工程先进技术译丛）

书名原文：FPGA - based Implementation of Signal Processing Systems

ISBN 978-7-111-55723-4

Ⅰ.①信…　Ⅱ.①罗…②李…　Ⅲ.①数字信号处理②可编程序逻辑器件 - 系统设计　Ⅳ.①TN911.72②TP332.1

中国版本图书馆CIP数据核字（2016）第311054号

机械工业出版社（北京市百万庄大街22号　邮政编码100037）
策划编辑：江婧婧　责任编辑：江婧婧　翟天睿
责任校对：刘雅娜　封面设计：马精明
责任印制：李　昂
三河市国英印务有限公司印刷
2017年3月第1版第1次印刷
169mm×239mm · 23.75印张 · 452千字
0 001—3 000册
标准书号：ISBN 978 - 7 - 111 - 55723 - 4
定价：109.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务	网络服务
服务咨询热线：010 - 88361066	机 工 官 网：www.cmpbook.com
读者购书热线：010 - 68326294	机 工 官 博：weibo.com/cmp1952
010 - 88379203	金 书 网：www.golden - book.com
封面无防伪标均为盗版	教育服务网：www.cmpedu.com

译 者 序

由于集成电路与 DSP 技术的迅猛发展，可编程逻辑电路的处理能力越来越强大，不仅具有强大的逻辑门阵列，而且嵌入了 ARM 核等微处理器与乘法器等协处理器，可以在 DSP、控制等相关领域广泛应用。国内外对可编程逻辑设计的专业人才需求较大，尤其是高级人才缺口较大。目前大部分可编程逻辑电路的相关教材着重讨论应用实现方法，本书则尝试将电路内部实现原理与方法呈现给读者，从而能从容应对高级应用开发。本书将 DSP 与 FPGA 逻辑电路原理与实例结合起来讲解，能很好地帮助开发者深入理解 FPGA 的原理并快速完成开发任务。

本书的一个重要特点是把 DSP 与 FPGA 结合起来讲解。从 DSP 逐步过渡到 DSP 算法的逻辑电路实现，使得采用 FPGA 电路实现 DSP 算法概念清晰化。

本书的另一个特点是详细比较了 Xilinx、Altera、Lattice 等大型 FPGA 生成时的电路实现架构，并指出了各种型号 FPGA 的特点，为开发者深入理解 FPGA 的原理、合理进行 FPGA 的选型有很大益处。

本书讲解深入浅出，适合有一定电路与信息处理基础的研究生或工程开发人员使用。本书可以帮助他们切实掌握信号处理的 FPGA 实现原理，并快速利用 FPGA 实现各种信号处理算法；还可以帮助他们深入理解 FPGA 架构，提升根据各种技术指标进行 FPGA 优化的基本技能。

全书共 14 章，由北方工业大学白文乐副教授、李争平副教授、黄明讲师以及北京邮电大学曾勇波博士进行翻译和校对。其中，曾勇波负责第 1 ~ 4 章、第 6 章和第 12 章；李争平负责第 5 章、第 7 ~ 9 章和第 13 章；黄明负责第 10 章、第 11 章和第 14 章。白文乐副教授对全书进行了统稿。

由于译者水平有限，译文不妥之处在所难免，希望广大读者批评指正。

译者

2016 年 10 月

原书序

数字信号处理（DSP）用途广泛，如高分辨电视、移动电话、数字音频、多媒体、数字相机、声纳探测器、生物图像、全球定位、数字无线电、语音识别等。在需求的不断推动下，DSP 技术在以上领域的应用直到硅芯片技术发展起来才成为可能。为以上应用开发可编程 DSP 芯片和专用片上系统解决方案在过去30 多年来一直是研发的热点。实际上，一类专用微处理器已经向 DSP 演进，即DSP 微处理器（或 DSPμs）。

硅技术成本逐渐增加，让开发专用片上系统（SoC）承受了相当大的压力，并且意味着这一技术将更多地面向高容量或专业化的市场。另一策略是使用微处理器类解决方案，比如微控制器、微处理器和 DSP 微处理器，但是有时候这些方法不能很好地满足许多 DSP 应用对速度、面积和功耗的要求。最近，现场可编程门阵列（FPGA）被认为是一种 DSP 硬件实现技术，因为它采用与 SoC 类似的方法，可以根据计算、存储和功耗的应用需求开发出最合适的电路架构。这一观点消除了 FPGA 只是“胶合逻辑”平台的偏见，且揭示了 FPGA 是多种组件的有机统一体，可以用来开发 DSP 系统。由 DSP 系统描述到实现高效的 DSP 系统仍然是一个十分复杂的问题，然而 FPGA 预制构件技术避免了开发 SoC 时面临的大量深层次细节问题。

本书从多层次对 FPGA 实现流程进行讨论，从而说明如何采用 FPGA 技术实现 DSP 系统。首先，本书包含了电路级优化技术，允许底层 FPGA 结构以逻辑资源中的查找表（LUT）与触发器形式被智能调用。对具体 DSP 算法运算的仔细研究表明，将系统需求映射为面积更加优化且更快速的底层硬件实现是可行的。这表明基于 LUT 的高效 FPGA 实现可以展现一些 DSP 系统的本质特性，如 DSP 变换、快速傅里叶变换（FFT）、离散余弦变换（DCT）和固定参数滤波器。

其次，考虑了从 SFG 表示创建高效的电路架构的问题。很明显，有效使用底层资源来匹配吞吐量的电路架构是成本最低的解决方案。这要求开发者挖掘DSP 系统高度规律的、高度计算的、数据独立的本质来为 FPGA 实现设计高度并行的流水线电路架构。多重分布式逻辑资源和专用的寄存器使得这种方法具有高度的吸引力。书中讲述了采用必要层次的并行与流水线为拟建设的系统搭建高效电路架构的技术。最后，随着技术演变，FPGA 现已形成了一个包含多种硬件和软件组件及互连构造的异构平台。很明显，人们对真正系统级设计流程的渴望需要更加高级的系统建模语言来支撑。前几章详述了在实现硬件功能时如何进行系

列优化的语言与方法，同时讲述了诸如互连与存储的系统级考量。这些目前仍然是很热门的研究方向。

本书涵盖了 FPGA 实现的这三个领域，并聚焦于后两个领域，即电路架构的创建和系统级建模，因为这些代表了更新的挑战。此外，电路级优化技术也在其他多处有非常详细的论述。这体现了本书和其他讲述 DSP 系统 FPGA 实现文献之间的一个主要区别。

总之，本书尽量以作者搭建 DSP 系统的实际经验来支撑论述。书中详细列举了大量的例子，其中包括在自适应波束形成器设计举例中基于 QR 的递归最小二乘（RLS）滤波器的实现，也包括自适应差值脉冲编码调制（ADPCM）语音压缩系统的设计。本书始终使用有限冲激响应（FIR）及无限冲激响应（IIR）滤波器来讲述映射和重定时。采用了一个基于 FFT 的应用讲述低功率优化，采用了波形数字滤波器（WDF）讲述分层重定时开发。

除了建模和设计方面，本书也关注知识产权（IP）核的发展，因为它已经成为 DSP 系统开发中一个关键的方面。当缺少相关的高级设计工具时，设计者可以依靠创造可重复使用的组件模块来缩小设计差距，即技术和设计师有效使用该技术的能力之间的差距。有一章重点描述了这种 IP 核的创建，另一章专门讲述了一个重要的自适应滤波器架构的核的创建，即 RLS 滤波器。

读者

本书针对致力于使用 FPGA 技术实现信号处理应用的工程师们。前面的章节力求帮助大学生和毕业生完成他们的学业，带领大家浏览 DPS 系统映射到FPGA 硬件时所做的各种权衡的简单例子。本书的中间部分包含了大量复杂的使用 FPGA实现 DSP 系统的实例，清楚展现了使用 FPGA 的好处。这些例子包括：矩阵乘法、自适应滤波系统、波形数字滤波器和基于 RLS 滤波器的自适应波束形成系统。从算法复杂度到 FPGA 硬件映射的清晰论述将为各类读者呈现利用 FPGA 硬件实现这些解决方案的专业知识，这在现有的文献中是无法找到的。值得一提的是，本书是作者 30 多年实践经验的完美归纳。

本书聚焦于可以构建复杂 DSP 系统的 FPGA 异构平台。需要特别说明的是，我们采用系统级的方法来解决问题，比如系统级优化、IP 核集成与实现、系统通信架构与低功耗实现等。目的是让设计者能够利用书中讲述的技术与实例，结合现有的基于 C 语言或 HDL 语言的开发方案实现他们自己的想法。

组织方式

本书的目的是通过 FPGA 技术实现 DSP 中具有挑战性的实例来启发人们；为了这一目的，它主要讲述把 DSP 算法转换为合适电路结构的高层映射，而不是赘述 FPGA 的具体优化。其他文献更多侧重于结合基于 HDL 的设计工具来讲述，而本书把 FPGA 当成可以开发复杂 DSP 系统的硬件来讲述。这样，FPGA 就成为

一个包含复杂资源的异构平台，把软、硬件处理器，专用DSP模块，处理单元通过可编程快速专用连接互联。本书由4个主要部分组成。

第一部分，即第2~5章讲述了DSP系统和实现的基础知识，帮助进入这些领域做入门引导。第2章从对DSP简单论述开始，涵盖了数字滤波和变换。详述了基本的滤波器结构，自适应滤波器算法。至于变换方面，本章简单讲述了FFT、DCT和离散小波变换（DWT）。同时，讲述了一些在心电图（ECG）的应用实例来说明一些关键知识点，为本书后续的例子做背景铺垫。

第3章重点讲述计算机运算，它在DSP系统实现中很重要。本章从对数字系统的思考与基本运算开始，引出了加法器和乘法器。这些是FPGA核心模块的代表，但主要思考的是二次方根和除法执行电路，因为一些DSP应用中需要这些知识。对其他数的表示也做了简单介绍，即有符号数的表示（SDNR）、对数系统（LNS）、余数系统（RNS）和坐标旋转数字计算机（CORDIC）。但是，因为没有例子使用这些数值系统，所以没有详细讲述。

第4章涵盖了各种实现DSP算法的现有技术。理解其他技术的特点很重要，有助于用户选择最合适的技术。在FPGA技术与其他技术比较中发现区别显著。这些技术包括以ARM处理器与DSP为代表的微处理器技术，主要详细讲述了德州电气TMS320C64系列。之后介绍了并行机制，包括脉动阵列、单指令多数据（SIMD）及多指令多数据（MIMD）。紧接着讲述了SIMD机制的两个例子，即Imagine处理器和Clearspeed处理器。

在这部分的最后，也就是第5章，详细讲述了商用FPGA，主要集中于两大供应商Xilinx和Altera的Virtex和Srratix系列FPGA，也讲述了一些Lattice、Atmel和Actel的技术。这一章给出了它们的架构细节、DSP的具体处理能力、存储器结构、时钟网络、互连框架和I/O及外部存储接口。

本书第二部分讲述了三个主要阶段的系统级实现，即从电路架构到具体FPGA系列上的有效实现、从信号流图（SFG）表示创建电路架构，以及来自一个计算表示高级模型的系统级规范和实现方法。在这部分的起始部分，即第6章，从电路架构描述角度讲述了FPGA设计的有效实现。因为这部分内容已有大量文献出版，所以这一章只是综述了现有的有效DSP实现技术，重点讲了分布式算法（DA），同时详细讲述了别处未讲到的的折减系数乘法器（RCM）。后边讲到的技术对固定参数函数很有用，如固定系数滤波器、DCT等变换。这一章也简要探讨了如存储器实现和延时实现的细节设计问题。

第7章概述了实现快速设计的工具，并用Petri网络和其他计算模型（MoC）的形式讲述了高级嵌入式系统的系统规范。涉及的工具有Gedae、Compaan、ESPAM、Daedalus和Koski。这一章也讲到了FPGA的IP核生成工具，包括Labview FPGA、Synplify DSP、基于C的快速IP核设计工具及MATLAB。

第8章讲述了下一个阶段工作，即DSP算法如何以信号流图（SFG）或者数据流图（DFG）的形式被映射到第6章开头讲述的电路架构中去。这个工作基于由K－K－Parhi撰写的一本优秀教材（《VLSI数字信号处理系统：设计和实现》，Wiley，1999），它讲述了大量技术如何被应用到基于VLSI的信号处理系统中。这一章讲述了DFG描述怎么被转换成不同级别的并行和流水线结构，形成最符合应用需求的电路架构。这部分技术以简单的FIR和IIR滤波器为例进行了讲解。

第9章介绍了IRIS工具，开发这一工具主要为了从DSP系统SFG描述和算法及第8章所述的大量特征中提取创建电路架构的过程产物。它用WDF例子进行说明，并展示了系统级设计中分层这个主要问题，提出白盒方法是一个可行的解决途径。这些章节为书中后续章节系统级问题的讲解做了准备。

本书的最后部分，第10章和第12章讲述了设计挑战的第三方面，突出讲述了高级设计。第8章和第9章已经讲述了如何提取DSP功能层来生成FPGA实现，通常被认为是生成DSP IP核的有效方法，是系统部分性能的标志。第10章详细讲述了创建硅IP核的原理，突出了不同的风格，即硬、软和固件风格，阐述了复用设计核心理念，这被认为是缩小设计效率差距的关键方法。IP核的产生已经成为长期与FPGA关联的增长型行业；实际上，以最短的设计时间获得高度有效的FPGA解决方案在使用FPGA实现DSP过程中变得至关重要。第10章讲述了以公司的实践经验为基础的核的产生细节，以及与之伴随的IP核演进历史，并回顾了如何创建可参数化IP核的整个过程，顺便简述了来自Xilinx和Altera的现有FPGA IP核。

沿着高级设计这条主线，第11章讲述了异构FPGA的模块化设计。这一章集中讲述数据流建模这一适合开发DSP系统的平台，介绍了多种风格的数据流，包括了同步数据流、环形的静态数据流、多维的同步数据流。快速综合和优化技术用以由DFG创建有效的嵌入式软件方案，涵盖了多个主题，如图表平衡、群集、代码生成和DFG作用物的可配置性。然后，这章描绘了如何经由白盒概念把流水线IP核包含进来，这里用了两个例子进行说明，即一个标准化的桥形网格滤波器（NLF）例子和一个固定的波束形成器例子。

第12章剖析了创建软的、高度可参数化的RLS滤波器核。这一章开始介绍了自适应波束形成和基于QR识别算法进行波束形成的有效方法。然后，从算法讲述开始，清楚讲述了如何由一系列架构形成单一通用架构。同时，考虑了定点和浮点算法的选择及控制开销等问题。第13章讲述了FPGA实现的关键部分，也是硬件设计的另一关键部分，即低功耗设计。只有在与微处理器比较时，FPGA才是所谓的低功率解决方案，在与相应的ASIC比较时，FPGA实现还存在很大差距。本章开始主要讲述了各种静态和动态功耗源，之后提出了大量的解决方案，首先来减少静态功耗，受限于FPGA的固有属性而减少有限，然后来减少

动态功耗，主要通过减少 FPGA 实现中的开关电容达到目的。本书使用了一个基于 FFT 的实现来说明在减小功耗中取得的一些收获。

最后，第 14 章归纳了本书中的主要方法，并考虑了将来可能被采用的FPGA架构。此外，它简述了一些在书中未包括的主题，特别是可重配置系统。通常认为 FPGA 的一个优势是在启动时可编程，在运行周期期间允许修改。但是值得关注的想法是，动态重配置 FPGA，允许在运行期间进行修改，即动态重配置（前边的模式可视为静态重配置）。其优势在于 FPGA 可以被当成虚拟硬件，利用可用硬件实现远超现有 FPGA 设备可用容量能够实现的功能。这是一个极具诱惑力的想法，但实际情况限制了其可行性。

致　　谢

作者有幸得到了大量同事、同学和朋友的帮助、支持和建议。作者真心感谢 Richard Walke 和 John Gray 在 Queen's University Belfast 给予 FPGA 方面大量启迪灵感的帮助工作。其他很多人也通过各种方式提供了技术或其他支持。他们是 Steve Trimberger，Ivo Bolsens，Satnam Singh，Steve Guccioe，Bill Carter，Nabeel Shirazi，Wayne Luk，Peter Cheung，Paul Mc Cambridge，Gordon Brebner 和 Alan Marshall。书中所述作者的研究经费有很多资助来源，包括工程和物理科学研究局、国防部、国防技术中心、奎奈蒂克公司、英国航空公司、Selex 和北爱尔兰的教育和学习机构。

有几章是与以下同事和同学共同创作的，他们是 Richard Walke，Tim Harriss，Jasmine Lam，Bob Madahar，David Trainor，Jean - Paul Heron，Lok Kee Ting，Richard Turner，Tim Courtney，Stephen McKeown，Scott Fischaber，Eoin Malins，Jonathan Francey，Darren Reilly 和 Kevin Colgan。

作者感谢 John Wiley & Sons 的 Simone Taylor 和 Nicky Skinner，他们无私奉献，提供了很多帮助，准备并协助完成了本书。

最后作者想感谢朋友和家庭的支持，他们是 Pauline，Rachel，Andrew，Beth，Anna，Lixin Ren，David，Gerry 和 Outlaws，Colm 和 David。

目　　录

第1章 FPGA概述

1.1 引言

电子技术的发展为20世纪带来了革命性的剧变，并且其产生的影响持续至21世纪。计算机工业的诞生和随后的发展、移动电话的发明、数字电视和无线业务的普及等都是电子技术发展的典型代表。20世纪70~80年代，电子系统开始集成到微处理器和带有数字逻辑器件的存储芯片等标准部件上，比如专用集成电路和具有专用输入/输出（I/O）器件的印制电路板（Printed Circuit Board，PCB）等。随着集成度的增加，制造PCB变得越来越复杂，很大程度上是由于晶体管和I/O引脚数量增加从而导致器件复杂度增加，同时，多层板发展到多达20个独立层。然而，器件连接的错误概率也同时增加，尤其是在产品发布前成功的设计和测试工作系统所允许的时间越来越短。

随着电路板技术的发展，系统描述难度增加，上面的问题变得更加严峻。开发新系统以满足演进标准的压力，或者由于系统变化、设计规范的变化导致电路板构建之后可能的改动等，都意味着物理系统构建和处理器软件代码开发方面的“完全定制”思想已经越来越不现实。微控制器和微处理器等可编程处理器的应用虽然能够让设计者在一定程度上对产品的系统进行校正或修改，但可做的改动有限，因为对PCB上器件的连接改动仅限于处理器I/O口的连接。因此，采用可编程连接或者“胶合逻辑”具有非常大的吸引力，现场可编程逻辑尤其是现场可编程门阵列（FPGA）技术由此诞生。

1.1.1 FPGA

现场可编程门阵列（Field Programmable Gate Arrays，FPGA）刚开始只是一种简单的“胶合逻辑”技术，为主要器件提供可编程连接，这种可编程性基于反熔丝技术、可擦除可编程只读存储器（Erasable Programmable Read Only Memory，EPROM）技术或静态随机存取存储器（Static Random Access Memory，SRAM）技术（Maxfield 2004）。此方法可以纠正那些在开发末期才发现的设计错误，大多是对FPGA重新编程，从而可以按照需求来改变器件之间的连接。虽然连接重新编程会增加项目开发时间，但它避免了成本高且耗时长的电路板重新设计，并且可显著降低设计风险。

与很多其他电子行业一样，FPGA 市场的形成和发展也受摩尔定律（Moore 1965）驱动，如图 1-1 所示。摩尔定律认为晶体管的数量每隔 18 个月翻一倍。这种难以想象的增长速度开辟了一片新兴市场，并且成为很多电子产品市场的强大驱动力，比如移动电话、数字音频设备、数字电视等。这是因为晶体管的数量不断翻倍的同时成本并没有增加，从而在每次技术革新时降低了单个晶体管的成本。技术的发展使得 FPGA 市场在 20 年间从零发展成集成电路产业的重要角色，目前的市场规模预计约 40 亿美元。

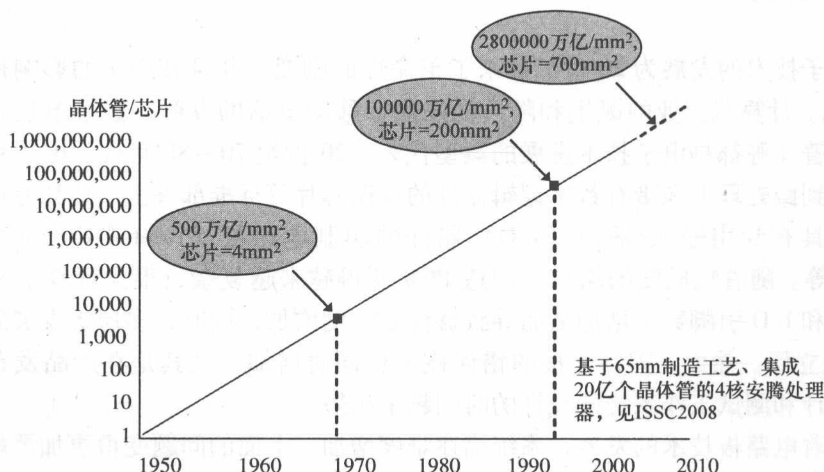


图 1-1　摩尔定律

很多时候，一些人根据摩尔定律提出的增长速度认为晶体管已经接近零成本，因此可以进一步将晶体管用在可编程硬件上，以便提供更多的灵活性。这一观点看似可以从成本变化的角度来实现：晶体管成本从 1980 年前后大约 0.1 美分降到 2000 年前后大约 0.001 美分。这些人认为这个观点也已经被实现，以 FPGA的形式将硬件可编程性引入电子产业就是证明。为了在 SRAM 技术中实现单个晶体管的可编程性，需要在晶体管的基极存储“1”或“0”来控制可编程能力，使其能够导通或者截止；然后将这个值存储在一个由 6 个晶体管组成的 SRAM 单元中，显然，引入可编程能力会导致晶体管数量增加 600%。虽然从完整的 FPGA 实现来看并没有什么影响，但设计中必须要考虑最终的系统成本。

FPGA 技术最大的吸引力在于 FPGA 硬件在加工后仍然可以进行编程修改的能力，在竞争日益激烈的市场中，“一次成功”的系统构建越来越难实现，但它提供了一种新的保障。20 世纪 90 年代末到 21 世纪初，市场处于低迷状态，当其他微电子技术都受影响的时候，FPGA 市场仍然保持稳定，证实了人们对 FPGA 市场发展的预判。这种情况可能会再次出现。当然，可编程能力的重要性早已经在微处理器中显现，但 FPGA 可编程能力代表了一种新的能力。

1.1.2 可编程能力和 DSP

上一节介绍了 FPGA 技术在可编程能力方面的优势，即可以降低制作 PCB 时存在错误的风险或者是对已生产的产品进行后期更改。虽然在 FPGA 技术发展初期，这个优势很突出，但半导体硅技术的发展，促使 FPGA 从可编程互连技术变成一个系统部件。如果微处理器或者微控制器被视作可编程系统部件，那么目前的 FPGA 器件也必须受到同等对待，让我们从不同的视角来审视系统的可编程能力。

在电子系统设计中，微处理器/微控制器主要的吸引力在于其通过降低设计复杂度显著地降低了系统开发过程中的风险。由于硬件是固定的，因此所有设计工作都可以集中在开发代码上——这些代码保证硬件能够按照系统规范工作。高效软件编译器的发展已经实现了这种推断，编译器大量地减少了设计人员编写汇编语言的需求。它还在一定程度上将设计人员从微处理器架构的细节中解脱出来（尽管很多实践者认为掌握微处理器的架构知识对写出好代码非常重要）。这种设计思想越来越受欢迎，因此嵌入式微处理器课程已经成为电子电气或计算机工程相关专业的必修课程。

很多处理流程已经归结到软件开发人员利用基础的处理器架构的能力，即冯·诺依曼架构。然而这种优势同样也成为其应用到本书所讲的数字信号处理（Digit Signal Process，DSP）的限制因素。在冯·诺依曼架构中，运算按照时间顺序处理，因此能够相对直接地从编程角度来解释硬件；然而，这种处理方式严重限制了 DSP 的性能，因为 DSP 的应用具有非常显著的并行化特征，并且其中大量运算与数据相对独立，从而具有优化的空间。DSP 应用需要并行实现，虽然 DSP 微处理器（也称为 DSP μs）在一定程度上通过同时采用并行硬件和软件“流水线技术”解决了这个问题，但仍然采用“一种架构来适配所有 DSP 问题”的思想。

FPGA 突破了这种限制，它将那些需要考虑的因素作为一种可编程能力，即对基础的处理器结构进行编程的能力。通过搭建能够尽量满足算法需求的架构来获得面积、速度和功耗等方面的最佳性能，这并不是一种新思想，设计一种满足算法需求的系统架构是专用集成电路（Application Specific Integrated Circuit，ASIC）的基础。大量的 ASIC 实现都是成本效益最高、最快和功耗最低的解决方案，但是它会增加掩模成本并且对系统实现“一次成功”率有影响，由此催生了更具吸引力的 FPGA。可以说，FPGA 既能达到 ASIC 实现的性能，同时又具有可编程处理器的可编程性优势。FPGA 从诞生到现在已经能在单个 FPGA 片上针对一些 DSP 应用实现每秒数千亿次的运算，其性能远远超过微处理器的性能。

1.2 节将结合芯片的发展过程来回顾硅半导体技术的演进，并且简要地介绍可编程能力的关键要素，1.3 节将详细讨论这些方面；1.4 节将介绍在利用 FPGA 技术的优点时所面临的挑战。

1.2 芯片发展简介

可能很多人认为 18 世纪末到 19 世纪初的工业革命对我们的生活和出行方式有重要的影响，也有一种观点认为半导体的发明对我们的生活具有同样的影响。半导体技术改变了我们与世界及相互之间的沟通方式，比如移动电话、电子邮件、视频会议等；我们可以通过电视、广播、数字录像参与娱乐活动；可以通过计算机和电子书开展教育；同样，还可以通过无线通信及计算机技术远程办公。

上面提到的这些技术都是在发明晶体管之后出现的，由贝尔实验室 William Shockley 的助手 John Bardeen 和 Walter Brattain 所发明。他们当时在研究半导体材料硅的特性，后来发现通过控制基极电压可以控制发射极和集电极之间的电子流。这一发现对电子技术具有重大影响，使得人们可以用更可靠的晶体管来替代真空管，并迅速推广应用在各种产品上。

另一个主要的演进发生在研制第一块硅芯片时，由 Jack Kilby 及 Robert Noyce 各自独立研制，验证了可以将器件集成在一块半导体材料上，即集成电路（Integrated Circuit，IC）。此外，Noyce 的方案解决了一些实际问题，使得集成电路的批量生产更简单。从设计和制造的角度来看，将晶体管和其他器件集成到一块芯片上具有很多优势。比如，不再需要通过人工布线的方式连接独立器件；电路可以做得更小并且可以采用自动化的制造过程。芯片技术的演进推动了由德州仪器（Texas Instrument，TI）公司开发的标准 TTL 7400 系列器件的发展，也推动了很多基础电子配件构建单元的发展。那个时候，人们还未意识到这些芯片将自成一个标准。

1968 年，Bob Noyce 和 Gordon Moore 研制了第一个微处理器 Intel 4004，这是一个重要的创新。虽然现在 64 位微处理器集成了 550 万个晶体管且每秒能执行数亿次运算，而当时在 $12mm^2$ 的面积上只集成了 2300 个晶体管，但这仍然是重大的进步。其重要的特征是通过改变微处理器中寄存器的代码来调整功能，并不需要重新做一套硬件平台。这对工程师非常重要，他们可以从部件的构建设计中解脱出来，如果这些部件不能轻易改动，那么就无法采用可以通过改变代码来增减功能的可编程平台。1965 年底的时候，Gordon Moore 在一篇文章中提出一个著名的预测，即摩尔定律（Moore’s law）。最初的论述是说最小部件成本下的复杂度大致每年翻一倍，后来修改为每 18 个月翻一倍。最具有代表性的是硅芯片技术的发展，使人们应用晶体管时不仅可以提供处理数据的能力而且可以通过简单地增加开销来提供可编程能力。虽然这意味着我们可以以极低的成本使用晶体管并且微处理器将成为主导，但问题是我们没有有效地利用这些晶体管。付出的代价是功耗增加了总体价格，进而影响系统的综合性能。在微处理器系统中，

仅有一小部分晶体管在运算中发挥作用。

在这个阶段，设计过程中的一个重要变化是芯片设计向广泛的爱好者打开了大门，包括大学生（本书的主要作者就是在当时开始接触芯片设计的）。Mead和Conway（1979）写了一篇经典的文章，显著地简化了芯片设计规范，使得小型芯片的实现甚至可以不需要检查设计规范。通过假设一些最坏情况，可以在给定芯片尺寸的情况下自动创建很小的设计规范集。芯片设计过程从此不再显得那么神秘，随着软件工具的发展，各个公司能够为自己的产品开发专用集成电路。美国的MOSIS程序（Pina 2001）结合上面的设计方法，提供了一套机制，为在校大学生和研究生提供IC设计教学和实践的机会。后来欧洲芯片（Eurochip，即现在的Europractice，Europractice 2006）开发了相同的套件，全欧洲的大学都可以借此批量设计和生产芯片。然而由于人们越来越在乎“一次成功”的设计，导致固定成本（Nonrecurent Engineering，NRE）上升，从而压制了ASIC的思想。NRE主要体现在制造工艺中掩模的生产成本上；成本上升是由于缩减硅芯片尺寸所需要的更精细几何形状的掩模越来越贵（并且难度越来越大）。Zuchowski等人在2002年首次解释了这个问题（Zuchowski等2002），图1-2所示为生产ASIC掩模所需成本（NRE成本的一部分）的上升趋势。

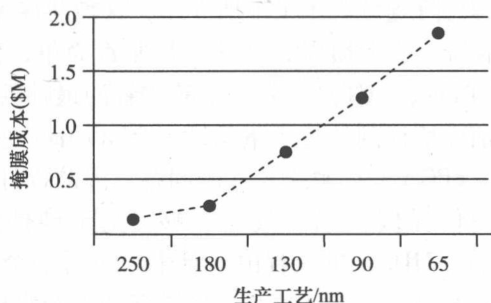


图1-2 不同技术阶段下的掩模成本（Zuchowski等2002）

1985年，Xilinx开发的XC2064 FPGA系列首次采用了FPGA的思想。与此同时，Altera公司也开发了一款可编程器件，后来被命名为EP 1200，它是第一款高密度可编程逻辑器件（Programmable Logic Device，PLD）。Altera公司的PLD器件是采用3μm的CMOS EPROM技术制造，需要紫外线来擦除程序；Xilinx的技术基于传统的静态RAM技术，需要一块EPROM来存储程序。Xilinx的联合创始人Ross Freeman认为，随着硅芯片技术的持续发展，晶体管将越来越便宜，从而可以用来提供可编程性能。这便是FPGA市场的开端，随后迅速在各大厂商中普及，包括Xilinx、Altera、Actel、Lattice、Crosspoint、Algotronix、Prizm、Plessey、Toshiba、Motorola和IBM。FPGA市场迅猛增长，Gartner Dataquest曾经指出2006年市场规模增长到45亿美元，2007年增长到52亿美元，2008年将增长到63亿美元。FPGA市场发生了很多变化，包括一次重大重组，很多厂商，比如Crosspoint、Algotronix、Prizm、Plessey、Toshiba、Moterola和IBM从FPGA市场退出；另外就是FPGA系列减少，以及因成本问题使得SRAM成为主流技术。FPGA市场目前由Xilinx

和 Altera 主导，更重要的是，FPGA已经从简单的胶合逻辑部件发展成整套的系统可编程芯片，包括板载物理处理器、软处理器、专用 DSP 硬件、存储器和高速 I/O。

在 20 世纪 90 年代，功耗问题成为焦点，虽然那时候 FPGA 已经预示了门阵列市场的结束，但 ASIC 仍然被认为具有广阔的市场空间，比如在以高性能和能量效率为驱动的移动通信市场上。各个厂商都用图表来比较 FPGA 和 ASIC 的性能，以吸引开发者选择他们的产品。然而，这种方法过于简单，对开发者的选择没有多大影响，因为在过去十年涌现了大量的其他技术，我们将在 1.2.1 节进行介绍。

Steve Trimberger 在一次大会演讲（Trimberger 2007）中巧妙地概括了 FPGA 的发展过程，表 1-1 对此做了总结。从表 1-1 中可以看到，FPGA 有三个不同的发展阶段。在“发明”阶段是 FPGA 的萌芽期，通常被当作系统部件来提供可编程的互连，如 1.1 节所介绍，这种互连可以为设计方案的进一步改善和变动提供保障。在此阶段，设计工具非常简单，但设计者仍然乐于挖掘查找表（Look Up Table，LUT）或者单个晶体管的最佳性能。在 20 世纪 90 年代初，出现了一次前面所提到的技术重整期，Trimberger 称之为“大架构调整”。在“扩张”阶段，FPGA 开始解决尺寸的问题，因此设计复杂度成为关键。意味着 FPGA 生产厂商仅提供布线工具还不够，基于硬件描述语言（Hardware Description Language，HDL）的流程由此诞生。最后一个是“聚焦”阶段，FPGA 开始整合处理器和高速互连。第 5 章将详细介绍 FPGA 的这些新特性。

表 1-1　三个时代的 FPGA

时间	时代	描述
1984 ~ 1991	发明阶段	技术有限，FPGA 远小于问题尺寸；设计自动化是次要的；架构效率是关键
1992 ~ 1999	扩张阶段	FPGA 大小接近的问题尺寸；设计便捷化成为关键
2000 ~ 2007	“聚焦”阶段	FPGA 远远超过了典型的问题尺寸；逻辑容量受限于 I/O 带宽

1.2.1　技术特点

除了 FPGA、ASIC 和微处理器，过去的 10 年中涌现了大量其他值得参考的技术。现介绍如下。

可重配 DSP 处理器这类处理器可以实现一定程度的定制，但仍然是基于固定的架构提供一些应用所需的功能。例如，Tensilica 的 Xtensa 处理器系列（Tensilica Inc. 2005）和 Elixent（即现在的松下）的 D－Fabrix，后者是一款可重配半导体知识产权（Intellectual Property，IP）核（Elixent 2005）。

结构化 ASIC 的实现，使得有人认为这是“门阵列”技术思想以结构化

ASIC 重新崛起，结构化 ASIC 是预定义的硅芯片框架，用户可以用缩减的硅芯片制造工艺来提供互连能力。这项技术由 Altera 通过硬拷贝技术来实现（Altera Corp. 2005），让用户可以将其 FPGA 设计迁移到 ASIC 上。

目前的状况是有很多技术共存面向不同的市场。本节已经简要介绍了硅芯片技术的发展过程，这些新技术的发展现在已经成为本书所讲的 DSP 电子开发系统的硬件。

一个更有意思的观点是考虑如何获得这些技术的可编程能力。图 1-2 所示为掩模成本和制造设备成本上升问题，使得针对应用于定制的解决方案发展前景黯淡。因此，专用芯片解决方案只适合大批量生产，只有大公司才能承担这种风险。有人认为纳米技术是可选的解决方案，但我们认为在最近 10 年内不太可能实现。结构化 ASIC 可以看成门阵列技术的重现（至少从技术原理看是这样），它将是低功耗应用的一种有吸引力的解决方案。然而，作者认为，其可编程能力预计要到下一代系统才能得到应用，届时，上市时间、生产成本和一次成功的硬件压力更重要，能够对硬件编程的思想将至关重要。下一节将从可编程能力角度来对各种技术进行对比。

1.3　可编程能力的影响

很多文献都用摩尔定律来说明芯片技术的发展。另一个有趣的观点特别适合说明 FPGA 技术的发展过程，即牧村浪潮（Makimoto's wave），首次发表在 1991 年一月份的电子周刊上。牧村次夫（Tsugio Makimoto）总结了这个规律，他发现半导体技术沿着“标准”和“定制”交替发展，如图 1-3 所示。在 20 世纪 60 年代初，人们开发了大量标准部件投入应用，即 TI 公司的 TTL 7400 系列逻辑芯片。20 世纪 70 年代初，开启了定制大规模集成电路的时代，人们为专门的应用定制和设计芯片，比如计算器。因为芯片集成度增加，开始出现中等规模集成电路（Medium Scale Integration，MSI）。微处理器在 20 世纪 70 年代的发展过程见证了芯片重新向标准化方向波动的过程，此时，一块“标准”芯片可以应用到很多场合。20 世纪 80 年代人们发明了 ASIC，设计者可以克服按序处理的微处理器的缺陷，从而解决了 DSP 应用中大运算量面临的运算速度问题。后来出现了 DSP 处理器，比如 TMS 32010，和传统处理器不同的是，它们基于哈佛架构，使用独立的程序和数据存储器及独立总线。即使有 DSP 处理器，ASIC 也提供了可观的处理能力，更重要的是功耗低。FPGA 从“胶合部件”发展而来，它可以将其他部件连接在一起形成一个系统，既可以是一个系统部件也可以自成一个系统，因此广受欢迎。将微处理器和 FPGA 耦合到异构平台上具有很大的吸引力，因为这是彻底的可编程平台，微处理器负责 DSP 系统的控制相关的部分，FPGA

负责数据相关的部分。这种思想构成了现场可编程定制计算机（FPGA - based Custom Computing Machines，FCCM）的基础，几大国际会议围绕该领域的课题定期召开，还为“可配计算”或“可重配计算”（Villasenor 和 Mangione - smith 1997）杂志奠定基础。在这些系统中，用户不仅能够在硬件上实现计算复杂的算法，还可利用硬件的可编程能力来改变系统的功能，即“虚拟硬件”，也就是说这些硬件可以虚拟地实现更大量级的系统（Brebner 1997）。第 14 章将继续讨论可重配系统。

我们认为可编程能力的发展曾经有两个重要阶段。第一个阶段发生在 20 世纪 70 年代微处理器出现的时候，此时工程师可以基于固定的硬件开发可编程的解决方案。当时主要的挑战是软件环境；即使已经有 C 语言的编译器和汇编程序，但开发者仍然使用汇编语言，通过手动编程的方法来达到最佳性能。于是人们开始设计库函数，提供最基本的 I/O 函数，从而让设计者能够集中精力在实现应用上。这些函数现在已经成为商业编译器和汇编程序的核心组成部分。随着对高级语言需求的增加，现在很多程序都通过高级语言，比如 C 语言和 Java 语言来实现，甚至更高级的语言环境，如 UML 也越来越多地被使用。

可编程能力发展的第二个阶段与 FPGA 有关。在图 1-3 中，牧村指出现场可编程能力在制造过程中被标准化，但也可以为应用定制。如果将第一个浪潮看成是软件领域的可编程能力而硬件保持不变，则此阶段可认为是提供了硬件可编程能力。这是一个很大的挑战，因为很多计算机编程工具主要是基于固定的硬件平台，可以做一些优化，主要是从算法方面来改进性能。通过 FPGA，用户可以自由定义，从而能够最好地适配应用的架构。然而也存在一个问题，即每种方案都需要人工去做，并且每个硬件工程师都需要知道设计的问题并验证硬件设计。

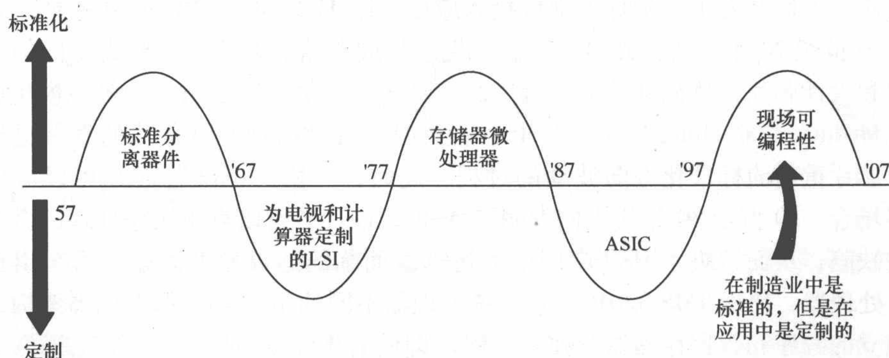


图 1-3 牧村浪潮（由 Reed Business Information 许可复制）

这两个阶段的一些趋势具有相似性。在早期，原理图捕获用于前期电路图设计，等同于汇编级的编程。随后出现硬件描述语言，比如 VHDL 和 Verilog，可

以用来生成更高层次的抽象，与目前使用的基于 C 语言的工具，如 Systemc 和 Mentor Graphics 的 CatapultC 作为基于单一软件的编程环境相似。起初作为软件编程语言时，人们有点不相信这种方法生成的代码质量。然而，通过开发改进成本 - 效率的综合工具，最终获得开发者的普遍认可。综合工具相当于高级编程语言软件编译器和库函数的演进，目前使用 HDL 来实现 FPGA 已经是常规方法。事实上，IP 核的出现反映了库函数的演进，例如，软件流的 I/O 编程函数常被当作通用函数进行复用，开发者都相信这种库函数代码的质量，因为随着技术的发展，他们面临着在相同时间内需要写出更多代码的压力。早期的 IP 核已经从基本库函数发展到复信号处理和通信函数，通常可从 FPGA 生产厂商和各种网络 IP 库获得。

1.4 FPGA 面临的挑战

在早期，FPGA 被当作胶合逻辑芯片将多个部件组合成一个复杂的系统。后来，FPGA 逐渐被当作完整的系统，见表 1-1。除了技术的演进，还有一些其他因素在推动 FPGA 的发展。比如，分布式算法（Distributed Arithmetic，DA）技术的应用推动了 FPGA 向 DSP 平台发展（Goslin 1995，Meyer - Baese 2001）。DA 技术使得FPGA的实现能够采用基于 LUT/加法器结构的 FPGA 模块，并且可以在一些 DSP 变换处理上获得相当可观的性能增益，比如定点系数滤波和快速傅里叶变换（Fast Fourier Transform，FFT）等变换函数。虽然这些技术说明 FPGA 可以为 DSP 应用提供高效的解决方案，但期待从 FPGA 硬件中尽可能挖掘性能，并且由几个人花数月时间来实现这种创新设计的想法已经不合时宜了。技术演进带来的复杂度增加，意味着当前 FPGA 技术所能提供的能力与设计者使用现有工具所能提供的解决方案之间的差距在扩大。这与 ASIC 产业中的“设计产出差距”（IRTS 1999）相似，即按照摩尔定律增长 60% 时，ASIC 设计能力却只增长 25%。由于设计者不需要处理亚微米的设计问题，因此这一问题在 FPGA 中并不严重。然而，一些关键问题仍然存在。

如何理解将 DSP 功能映射到 FPGA 上。有些方面是相对基本的，比如乘法运算、加法运算和延迟分别被映射到板载乘法器、加法器和寄存器及 RAN 单元。然而，理解浮点和定点、字长优化、FPGA 中算术变换的成本函数及路径延迟的影响等都是必须从系统层面考虑的问题，并且处理起来更加困难。

设计语言。目前，像 VHDL 和 Verilog 等硬件描述语言及其各自的综合过程都已经成熟。然而，随着复杂度的增加需要将固定和可编程微处理器核集成到一个完整系统，用户开始用审慎的眼光评判 FPGA，并且寻求能够更清晰地表示系统描述的设计方法。因此，EDA 领域有一种趋势是使用 C 语言作为设计语言，

但同时使用其他表示方法，比如类似同步数据流的基于计算模块（Model of Computation，MoC）的方法。

开发和使用 IP 核。当缺乏针对设计语言和综合问题快速可靠的解决方案时，片上系统（System on Chip，SoC）实现的 IP 市场应运而生，它填补了这个空当，并且能够对硬件进行快速原型设计。由于设计功能可以通过 HDL 实现，并且可以通过传统的综合工具高效地转化为 FPGA 代码，因此软核具有很大的吸引力。此外，人们已经开发了处理器核，从而可以增加专用的功能。这些方法的吸引力在于平台大部分是固定的，从而可以快速实现特定应用的功能。

设计流程。大多数设计流程的效率与通过高层实现的 FPGA 功能的描述方式有关，大多数是复数函数。现在，FPGA 技术发展速度很快，构成 FPGA 和处理器的系统开始以 SoC 平台出现，甚至直接用 FPGA 来实现单独的 SoC 平台，主要是因为 FPGA 具有板载硬核和软核、高速通信和可编程资源的优点，可以构成一个完整的系统。通常，软件流程比处理器甚至多处理器更先进，因为架构是固定的。虽然已经开发了 FPGA 等硬件平台的开发工具，但包括处理器和 FPGA 的异构平台，仍然需要软件来实现。

本书后续章节将详细探讨 FPGA 所面临的这些挑战。

参考文献

Altera Corp. (2005) Hardcopy structured asics: Asic gain without the paint. Web publication downloadable from http://www.altera.com.

Brebner G (1997) The swappable logic unit. *Proc. IEEE Symp. on FPGA-based Custom Computing Machines*, Napa, USA, pp. 77–86.

Elixent (2005) Reconfigurable algorithm processing (rap) technology. Web publication downloadable from http://www.elixent.com/.

Europractice (2006) Europractice activity report. Web publication downloadable from http://europractice-ic.com/documents_annual_reports.php.

Goslin G (1995) Using xilinx FPGAs to design custom digital signal processing devices. *Proc. DSPX*, pp. 565–604.

IRTS (1999) *International Technology Roadmap for Semiconductors*, 1999 edn. Semiconductor Industry Association. http://public.itrs.net

Maxfield C (2004) *The Design Warrior's Guide to FPGAs*. Newnes, Burlington.

Mead C and Conway L (1979) *Introduction to VLSI Systems*. Addison-Wesley Longman, Boston.

Meyer-Baese U (2001) *Digital Signal Processing with Field Programmable Gate Arrays*. Springer, Germany.

Moore GE (1965) Cramming more components onto integrated circuits. *Electronics*. Web publication downloadable from ftp://download.intel.com/research/silicon/moorespaper.pdf.

Pina CA (2001) Mosis: IC prototyping and low volume production service *Proc. Int. Conf. on Microelectronic Systems Education*, pp. 4–5.

Tensilica Inc. (2005) The Xtensa 6 processor for soc design. Web publication downloadable from http://www.tensilica.com/.

Trimberger S (2007) FPGA futures: Trends, challenges and roadmap *IEEE Int. Conf. on Field Programmable Logic*.

Villasenor J and Mangione-Smith WH (1997) Configurable computing. *Scientific American*, pp. 54–59.

Zuchowski P, Reynolds C, Grupp R, Davis S, Cremen B and Troxel B (2002) A hybrid ASIC and FPGA architecture. *IEEE/ACM Int. Conf. on Computer Aided Design*, pp. 187–194.

第2章 DSP基础

2.1 引言

在电子技术发展早期，信号以其自然的形态进行处理和传输，通常源信号（比如语音）产生模拟信号，然后转换为电信号，再通过合适的媒介（比如宽带）连接进行传输。人们很早就意识到DSP的优势。首先，数字硬件通常比模拟器件性能更好并且更稳定，而模拟器件容易老化，并且在生产过程中容易出现不确定的因素而影响性能。其次，DSP具有确定的精度，特别是它可以准确地复制信息（Rabiner 和 Gold 1975）。此外，通信界有强烈的意愿将多种信号传输网络，如电话网、地面电视网和计算机网络融合为单一的或者多种数字传输网，因此迫切需要将各种信息格式转换为对应的数字信息格式。

微处理器、DSP微处理器和FPGA都是处理数字信号的合适平台，所以了解诸如在FPGA平台上实现DSP算法面临的一些基本问题变得非常重要。这些问题范围很广，比如了解不同应用的采样速率和计算速率以便理解这些需求是如何影响最终FPGA实现的，再比如针对特定FPGA平台选择合适的数字表示形式及这种选择对DSP系统性能的影响。算法的选择和运算的精度会对最终实现的质量有深刻影响。

本章将介绍背景知识并解释上述问题。首先介绍DSP的一些基本思想，它们会影响硬件实现，比如采样速率、计算速率和延迟。然后简要地描述常用的DSP算法，先回顾各种变换，包括快速傅里叶变换（FFT）、离散余弦变换（DCT）和离散小波变换（DWT）。本章还将回顾滤波过程并简要介绍有限冲激响应（FIR）滤波器、无线冲激响应（IIR）滤波器和波形数字滤波器（WDF）。最后一节将重点介绍自适应滤波器，包括最小均方（LMS）算法和递归最小二乘法（RLS）。信号的数字化意味着如何表示和处理这些信号对最终系统的精确度非常重要，因此本书的最后一章将讨论DSP实现算法的算术意义。

本书主要针对DSP系统在FPGA硬件上的实现，因此本章将向读者介绍DSP算法，从而为后面将要描述的很多实例提供背景知识。参考文献列出了一些介绍DSP系统背景的入门书籍，其中既有基本原理（Lynn 和 Fuerst 1994，Williams 1986），也有更深入讨论的教程（Rabiner 和 Gold 1975）。此外，我们推荐Omondi关于计算机算术运算的书作为入门读物（Omondi 1994）。

本章的结构如下。2.2 节将详细介绍信号数字化的过程；2.3 节将介绍 DSP 的基本思想，特别是采样速率、延迟及流水线等与 FPGA 实现相关的问题；2.4 节将介绍 DSP 变换，包括 FFT、DCT 和离散小波变换；2.5 节将总结基本的滤波过程；2.6 节将描述自适应滤波算法。

2.2 DSP 系统基础

现在对信息处理、解释和理解的需求越来越强烈，在很多领域，如工业、军事和消费产品都有应用。这些应用需求涉及语音、音乐、图像或视频，也可能涉及通信系统中的错误检测、纠正及加密算法。因此需要对大量不同类型的内容进行实时处理，采样速率的范围可以从生物应用的几 Hz 到图像处理应用中的几十 MHz。很多情况下，信号处理的主要目的可能是增强有用信号部分，比如图像处理中的边缘检测；或者是消除干扰，比如雷达系统中人为干扰信号；也可能是移除不需要的输入，比如消除电话系统的回声或噪声。另一类 DSP 算法主要是用于捕获、存储和传输数据、音频、图像及视频，以及在数字广播和电信系统中成功应用的压缩技术。

近几年，很多类似上述处理的需求已经被标准化，图 2-1 所示为大量应用中所需要的算法。在通信系统中，为了使用正交频分复来提供高效传输，需要实现执行 FFT 的电路。在图像压缩中，联合图像专家小组（Joint Photographic Experts Group，JPEG）和后来的运动图像专家小组（Motion Picture Experts Group，MPEG）的演进推动了 JPEG 与 MPEG 标准的发展；这些标准涉及大量核心 DSP 算法，尤其是 DCT 和运动估计与补偿算法。

人们很早就意识到将信号数字化的优势，一方面是由于数字硬件通常在性能上和稳定性上都优于模拟器件，另一方面，模拟硬件容易老化并且在生产过程中质量不确定。DSP 则可以提供可靠的精度，尤其是能够完美复制（Rabiner 和 Gold 1975）。DSP 的迅速发展得益于成本不断降低的硬件发展，使得 DSP 技术能够简单地与计算机技术结合，在很多情况下还可以在相同的计算机系统上实现。图 2-1 中提到的很多应用增加了对复杂 DSP 系统的需求，反过来又促进了实现高效 DSP 算法研究领域的发展。这也同样促进了对 DSP 微处理器的需求，我们将在第 3 章介绍。

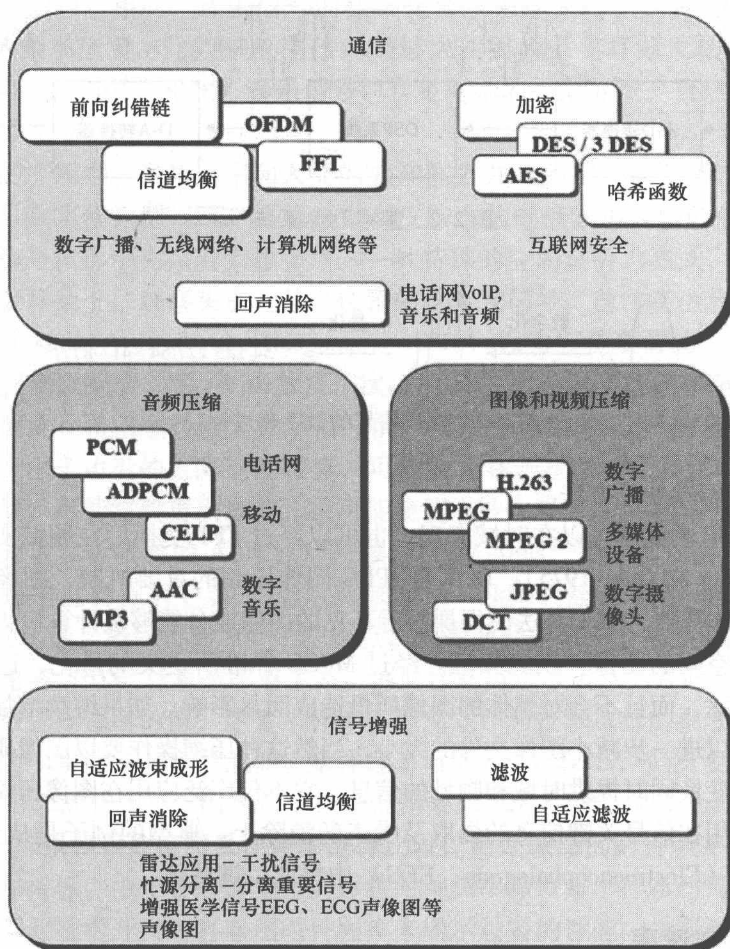


图 2-1 DSP 应用示例

2.3 DSP 系统定义

图 2-2 所示为 DSP 系统的基本实现过程，模拟信号首先经过模－数（A-D）转换器转换成数字信号，经过 DSP 系统处理后再转换回模拟信号。信号的数字化通过图 2-3 所示的过程实现，模拟信号首先转换为一组脉冲信号，然后量化为一组数字。通常将 DSP 系统的输入数字流记为 $x(n)$，输出信号记为 $y(n)$。原始模拟信号可能来自于声音、音乐、生物信号、无线信号、雷达脉冲或图片。显然，数据的表示方法是一个关键因素，本章后续章节将进行讨论。由于将信号数字化为数据的操作、存储或传输提供了多种可能途径，因此如图 2-1 所示，从而



图 2-2 基本 DSP 系统

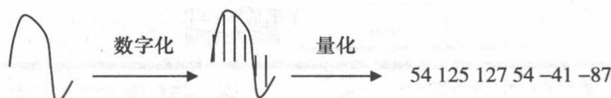


图 2-3 模拟信号的数字化过程

催生了各种信号处理方法。

很多 DSP 功能既可以在时域实现，也可以经过 FFT 变换后在频域实现，比如滤波（Rabiner 和 Gold 1975）。DCT 是 JPEG 图像压缩的重要机制，图像压缩又是 MPEG 标准的基础。DCT 算法使得裸眼看不见的图像成分能够被计算机识别，基本思想就是将空间的图像转换到频域。经过 MPEG 标准所定义的量化，这些图像成分可以被移除，而且不会对整体的图像质量造成明显影响。如果继续增加数据的移除量，则可以进一步减小图像文件的大小，当然这种压缩操作要以图像质量下降为代价。小波变换同时提供时域和频域的信息，它不仅广泛应用在图像压缩中，同时也可大量应用在信号关键信息的提取及噪声的消除上。典型的例子是从生理信号，比如脑电图（Electroencephalogram，EEG）中提取关键特征。

2.3.1 采样速率

奈奎斯特 - 香农（Nyquist-Shannon）采样定理是 DSP 系统最基本的一个原理。该定理指出：如果信号是有限带宽并且采样频率大于 2 倍信号带宽，则可从采样信号精确重构出连续时间基带信号。更详细的解释请读者参考香农和奈奎斯特的文章（Nyquist 2002，Shannon 1949），以及前面提到的一些教程（Lynn 和 Fuerst 1994，Rabiner 和 Gold 1975，Williams 1986）。简单地说，在将模拟信号数字化时，采样速率必须至少是最大频率 f_m（被数字化的这段信号）的两倍，才能够保全信息并且避免混叠（Shannon 1949）。换句话说，信号要求是有限带宽，意味着某个最大频率 f_m 以外没有频谱能量，从而可以确定奈奎斯特采样速率 f_s 为 $2f_m$。

举个简单的例子，对语音的标准采样速率是 8kHz，足以获得能够精确表示语音信号的频率分量，因为 4kHz（甚至可能是 3kHz）以上的频谱能量对信号质

量几乎没有影响。相反，音乐信号的数字化要求采样速率达到 44.2kHz 去覆盖 22.1kHz 的频率范围，此频率范围已经足够大，因为人类耳朵无法检测到超过 18kHz 的频率。与语音相比，音乐信号包含更复杂的频谱分量，自然就需要相应地增加采样速率。

在有的应用中，采样速率和人类的感知能力并无关系，需要考虑其他因素。以生物信号数字化为例，EEG 是通过贴在皮肤上的电极收集大脑的生物电信号，所捕获的信号波形中夹杂着大量噪声。一种特殊的应用是听力测试，将刺激物放置在受测者耳朵上，观察头皮上某个位置的脑电波信号。通常将该测试称为听性脑干活动（Auditory Brainstem Response，ABR），因为它是要寻找大脑的脑干区域的脑电图刺激响应，每 10ms 激发一次。ABR 波形有意义的频率范围是 100 ~ 3000Hz，因此在将信号转换为数字信号前，需要将记录的 EEG 信号进行带通滤波。然而，由于边沿的滚降速度较慢，可能会存在干扰频率。将 EEG 信号转换为数字信号，可能需要再次滤波，很可能使用小波去噪算法来移除上边沿和下边沿的频率分量。所需要的 ABR 波形持续时间为 20ms，刺激前后各 10ms。EEG 以 20kHz 频率进行采样，其奈奎斯特频率为 10kHz，超过信号最高频率分量（3kHz）的 2 倍。最后得到 200 个采样点，刺激前后各 100 个采样点。

采样速率与 DSP 计算速率有关系，因此性能需求可表示为吞吐速率。图 2-4 所示为典型采样速率的值，尽管采样率不能独立作为技术选择的参考。比如，音频应用中的 128 抽头 FIR 滤波器，采样速率可能是 44.2kHz，但所要求的吞吐速率将达到 11.2M 样本/s，因为每个样本都需要做 128 次乘法和 127 次加法（255 次运算）。如果错误地使用采样速率进行比较，则处理器的时钟频率也将同样错误地被用作指标。由于吞吐速率可通过将时钟频率除以每个样本所需要的处理周期得到，因此技术开发者喜欢用时钟频率来展示设备的性能。最典型的是，个人电脑（Personal Computer，PC）通常用时钟频率作为性能指标，但是如我们将在第 3 章将所要讨论的，吞吐速率更加重要。

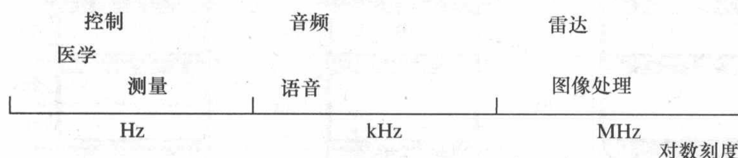


图 2-4　不同 DSP 系统中的采样速率

2.3.2　时延和流水线

延迟是指系统从接收到输入的时刻开始到产生结果为止所需要的时间。在同步系统中，延迟可以定义为产生输出之前所需要的时钟周期数。假设一个简单的

系统，输入为 a_0 和 $x(n)$，输出为 $y(n)$，用 T_{Latency} 表示其延迟，如图 2-5 所示，设 T_D 表示从输入到输出所需的时间，则吞吐速率可用 $1/T_D$ 表示。

在同步系统中，吞吐量与时钟速率有关。假设 T_D 表示数字电路中寄存器之间的关键路径，就可以确定可得到的最大时钟速率，并且和逻辑器件的数量及两个寄存器之间的物理距离有直接关系。通过增加额外的流水线操作阶段，可以减少电路中的关键路径，但其代价是增加了延迟及电路面积和功耗。图 2-6 通过简单的示例解释了流水线的过程。图 2-6 中，计算模块有 6 个独立的阶段。图中的流程描述了如何在电路的某些位置放置流水线阶段，从而减少寄存器之间的物理距离，最终增加时钟速率。图 2-6a 中，延迟为一个时钟周期，但最大时钟速率只有 36.4MHz。图 2-6b 中，每个主要的逻辑块之间加了流水线阶段，其最大时钟速率达到 100MHz，延迟为 6 个时钟周期。如果将一个流水线阶段移到其中一个较大的计算模块（比如模块 C），并且去掉模块 D 后面的流水线阶段，则如图 2-6c 所示，关键路径可以缩短到 5ns，时钟速率最大可达 200MHz，但仍然可保持延迟为 6 个时钟周期。图中的示例解释了电路时序优化的基本原理，它是高效设计过程和设计工具中的重要特征，第 8 章和第 9 章将详细讨论。

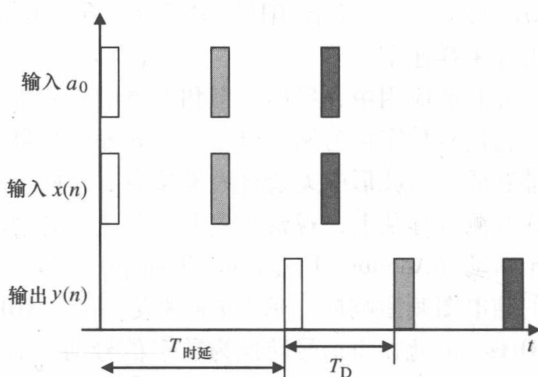


图 2-5 DSP 系统延迟示意图

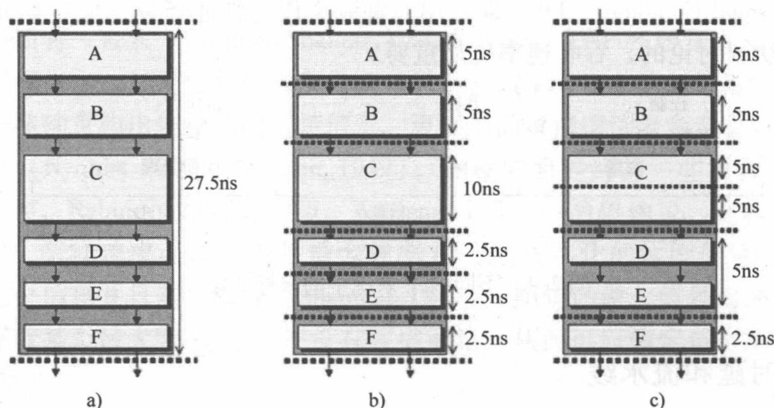


图 2-6 流水线简单示例

a）时钟速率 = 36.4MHz b）时钟速率 = 100MHz c）时钟速率 = 200MHz

2.4 DSP 变换

本节将简要回顾 2.2 节中提到的一些关键 DSP 变换，并简单介绍其应用。

2.4.1 快速傅里叶变换

傅里叶变换是将信号的时域表示转换为频域表示。基本思想是将信号分解成频率分量，将信号用一组正弦波和余弦波来表示。周期函数 $f(t)$ 的傅里叶级数展开可用式（2-1）表示。

$$f(t) = \frac{1}{2}a_0 + \sum_{n=1}^{\infty}[a_n\cos(\omega_n t) + b_n\sin(\omega_n t)] \tag{2-1}$$

式中，对任意非负整数 n，ω_n 表示 $f(t)$ 的第 n 次谐波，单位为 rad（弧度），记为

$$\omega_n = n\frac{2\pi}{T} \tag{2-2}$$

a_n 表示 $f(t)$ 的偶次谐波傅里叶系数，记为

$$a_n = \frac{2}{T}\int_{t_1}^{t_2}\cos(\omega_n t)\,\mathrm{d}t \tag{2-3}$$

b_n 表示奇次谐波傅里叶系数，记为

$$b_n = \frac{2}{T}\int_{t_1}^{t_2}\sin(\omega_n t)\,\mathrm{d}t \tag{2-4}$$

离散傅里叶变换（Discrete Fourier Transform，DFT）是连续傅里叶变换的离散形式，用于采样信号的变换。输入序列在时间上有限，DFT 仅计算这一段时间内信号的频率分量，因此离散傅里叶逆变换（Inverse DFT，IDFT）将只利用这些频率分量重构，可能无法完全重构原始信号（周期信号除外）。式（2-5）给出了 DFT 的定义，其中输入采样信号是复数序列 $x(0),x(1),\cdots,x(N-1)$，变换后的输出用复数序列 $X(0),X(1),\cdots,X(N-1)$ 表示。

$$X(k) = \sum_{n=0}^{N-1}x(n)\mathrm{e}^{-\frac{2\pi i}{N}kn} \tag{2-5}$$

式中，$k=0, 1, \cdots, N-1$。

FFT 是 DFT 的一种高效实现方法，对很多应用影响都非常大。其中一个应用是实现正交频分复用技术（Orthogonal Frequency Division Multiplexing，OFDM）。这种扩频数字调制方式应用在通信商，尤其是无线通信技术，比如，它大幅提高了 802.11 标准所能提供的数据速率。其中的算法就是基于快速傅里叶逆变换（Inverse FFT，IFFT）得到的频率分量正交的性质，从而可以让每个频

率分量作为一个子载波。值得注意的是，接收机使用 FFT[⊖]来检测子载波并重构传输信号。每一个子载波上仍然采用通常的低符号速率调制方式，比如相移键控或正交幅度调制，由应用来决定具体使用的调制方式。在 802.11a 中，数据速率最高可达到 54Mbit/s，实际速率取决于环境条件和噪声。如果噪声功率较大，则可以采用相移键控调制，此时数据速率较低；如果噪声功率较小，则可以采用正交幅度调制（Quadrature Amplitude Modulation，QAM）。图 2-7 所示为一个典型通信系统发射链路的主要组成部分。

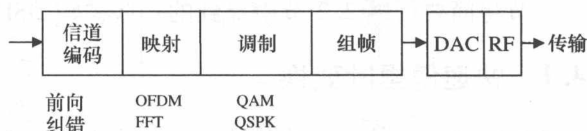


图 2-7　无线通信系统发射机

IEEE 802.11a 无线局域网标准使用 OFDM 可以在美国 5GHz ISM 频段 125MHz 带宽范围内工作，20MHz 的信道上使用了 52 个频率（子载波），其中 48 个用于传输数据，另外 4 个用于同步导频。同步导频非常重要，因为保持正交性是 OFDM 的基础，要求收发机精确同步。

2.4.2　离散余弦变换

离散余弦变换（Discrete Cosine Transform，DCT）以 DFT 为基础，但只针对实数，即 DFT 变换的余弦部分，其定义见式（2-6）。

$$X(k) = \sum_{n=0}^{N-1} \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right] \tag{2-6}$$

式中，$k=0, 1, \cdots, N-1$。

式（2-7）给出了二维离散余弦变换的形式，它是 JPEG 图像压缩的核心计算方法，也是 MPEG 标准的特点。

$$F_{u,v} = \alpha(u)\alpha(v)\sum_{x=0}^{7}\sum_{y=0}^{7} f_{x,y}\cos\left[\frac{\pi}{8}\left(x + \frac{1}{2}\right)u\right]\cos\left[\frac{\pi}{8}\left(y + \frac{1}{2}\right)v\right] \tag{2-7}$$

式中，u 表示水平方向的频率，$0 \leqslant u \leqslant 8$；$v$ 表示垂直方向的频率，$0 \leqslant v \leqslant 8$；$\alpha(u)$和 $\alpha(v)$是常数；$f_{x,y}$表示像素(x,y)的值；$F_{u,v}$表示（u，v）位置 DCT 系数的值。图 2-8 所示为 DCT 用于 JPEG 图像压缩的过程。示例中的图像大小为 8×8 像素，对图像的行和列实施离散余弦变换。频率解析结果将更重要的低频部分放在矩阵的左上角，沿着矩阵向其右下角移动时，频率分量增加。一旦图像全部转换为数值表示的频率分量，高频分量便可以通过量化过程被移除掉，因为这些分量对图像质量影响较小。当被移除的频率分量越多，压缩率越高，到一定程度

⊖ 应该为 FFT，原书有误——译者注。

时，图像质量将开始变差。我们称之为有损压缩。图像处理后的数值结果通过“之”字形方式输出，如图 2-8 所示。

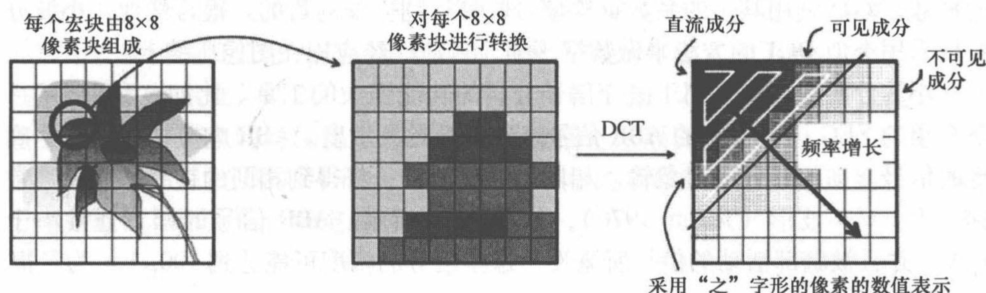


图 2-8　DCT 在图像压缩中的应用

2.4.3　小波变换

小波是一种快速振荡衰减的波形。小波分解是多分辨率滤波和分析的有力工具，通过将原始小波（母小波）缩放后的小波与输入信号相关来实现。小波变换将信号分解到频带上并且携带时间信息（Mallat 1989），对于准平稳波形的频率分析非常实用，而时不变 FFT 可能无法提供完整的信息。

目前，人们提出了很多种小波族，比如 Daubechies、Coiflet 和 Symmlet（Daubechies 1992）。小波分解可通过多种方式实现，即下一节将介绍的连续小波变换（Continuous Wawelet Transform，CWT）或离散小波变换（Discrete Wavelet Transform，DWT）。

2.4.4　离散小波变换

DWT 通过一组滤波器实现，如图 2-9 所示，图中描述了 6 层小波分解的示例。在 DWT 的每个阶段，输入信号都通过一个高通滤波器和低通滤波器，得到有关细节和逼近信息的系数。

式（2-8）是低通滤波器的表达式。在每个阶段都移除一半频率，信号的信息可以用一半数量的系数表示，因此，低通滤波器和高通滤波器公式可分别写为式（2-9）和式（2-10），其中，n 用 $2n$ 代替，表示以下采样过程：

$$y(n) = (xg)(n) = \sum_{\infty}^{-\infty} x(k)g(n-k) \tag{2-8}$$

$$y_{\text{low}}(n) = \sum_{\infty}^{-\infty} x(k)g(2n-k) \tag{2-9}$$

$$y_{\text{high}}(n) = \sum_{\infty}^{-\infty} x(k)h(2n-k) \tag{2-10}$$

小波分解是子带滤波的一种形式，在 DSP 中已经广泛应用。通过将信号分解到如图 2-9 所示的频率带，先去除表示一些高频分量（比如 D1 层和 D3 层）的系数，然后利用其余的系数重构信号，可达到去噪的目的。很自然地，小波分解也采用类似 DCT 的方法来做数据压缩，目前已经应用在图像压缩上。

小波分解同样也是用于医疗信号分析的功能强大的工具。比如，利用 2.3.1 节介绍的 EEG 记录得到的 ABR 信号来确定听觉灵敏度。ABR 响应本身是一个确定的信号，即给相同的对象输入相同的激发信号，将得到相同的锁时响应，通常称之为 Jewett 波形（Jewett 1970），如图 2-10 所示。ABR 信号的幅度通常小于 1μV，并且被脑部活动的信号所淹没，这些信号的幅度可能达到 100μV。为了提

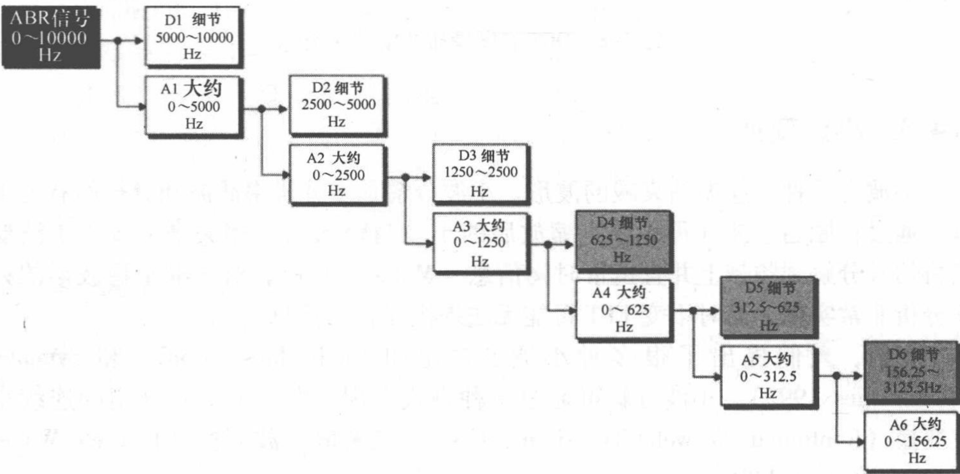


图 2-9 6 层离散小波分解示意图

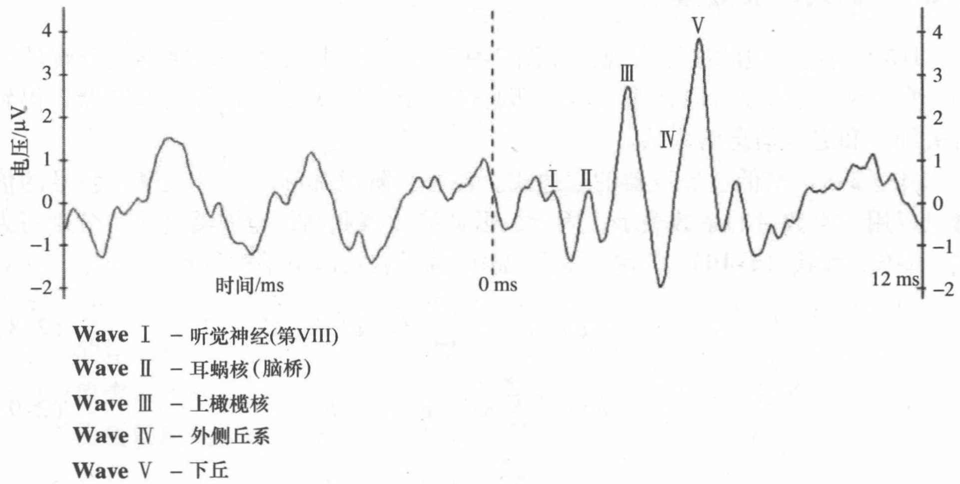


图 2-10 EEG 信号呈现的 ABR Jewett 波形

取 ABR 信号，常常将几千个响应信号平均。由于背景噪声本质上是白噪声，因此平均之后使得噪声趋于 0。同时，保留了确定性的 ABR 信号，波形可能稍微有些变化。即使抑制了噪声，但仍然很难判断被测对象是否听到激发声音，尤其是当激发声音强度正好处于门限附近时。响应锁定在激发后 10ms，并且，在某些频带有尖峰出现，即 200Hz、500Hz 和 900Hz 位置。小波分解能够在提取出这些关键的频率信息同时保留一些有用的时域信息来隔离这些峰值区域。图 2-11 比较了激发前后波形的 D4 带（见图 2-9）的小波系数。图中形象地解释了小波分解是如何聚焦到信号特定的时频区域的。

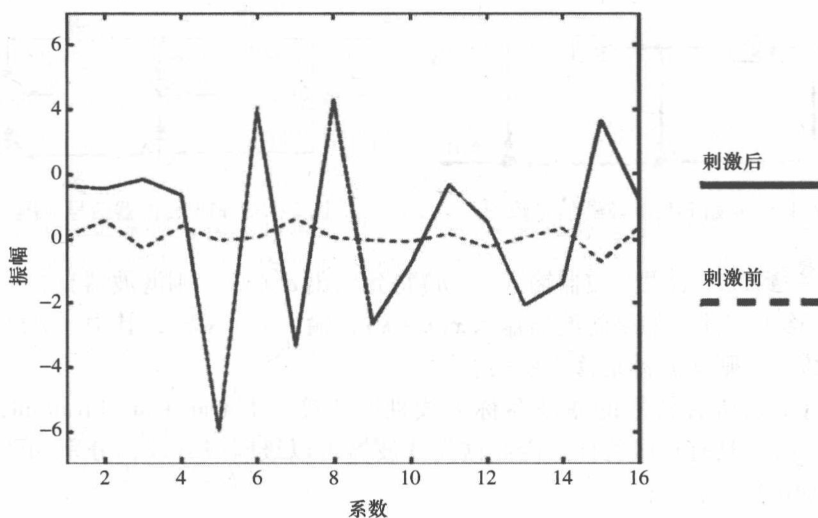


图 2-11 D4 层小波分解系数

2.5 滤波器结构

2.5.1 有限冲激响应滤波器

式（2-11）是一个简单的有限冲激响应（Finite Impulse Response，FIR）滤波器的表达式，式中 a_i 表示生成低通或高通滤波器响应所需的系数，N 表示滤波器抽头的数量。通常 a_i 和输入 $x(n)$ 长度都为 N，即对 $x(n), n=0,1,\cdots,N-1, a_i$ 不为零。

式（2-12）的函数式可以通过经典的信号流图（Signal Flow Graph，SFG）来表示，如图 2-12 所示，其中 $N=3$。在图中，延迟模块 z^{-1} 表示数字延迟，数据通过不同的分支输送，带标签的分支表示要与变量相乘，黑点表示求和。当乘

法和加法用功能模块表示时，也可以用图 2-13 所示的模块图来描述。

$$y(n) = \sum_{i=0}^{N-1} a_i x(n-i) \tag{2-11}$$

$$y(n) = a_0 x(n) + a_1 x(n-1) + a_2 x(n-2) \tag{2-12}$$

FIR 滤波器的一些重要特点如下。

1）叠加性。当满足如下条件时，叠加定理成立：如果滤波器输入为 $x(n)+u(n)$，则输出为 $y(n)+v(n)$，而 $y(n)$ 和 $v(n)$ 分别是从输入 $x(n)$ 和 $u(n)$ 得到的。

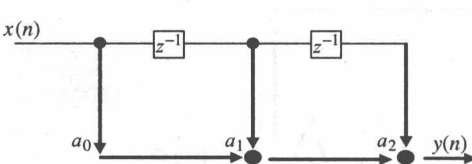


图 2-12　原始 FIR 滤波器信号流图

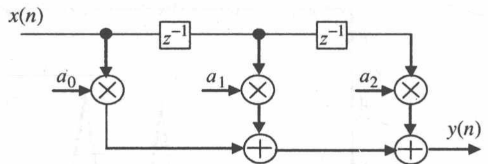


图 2-13　FIR 滤波器信号流图

2）一致性。如果滤波器输入 $a_x(n)$ 得到输出 $a_y(n)$，则滤波器具有一致性。

3）移不变性。如果滤波器输入 $x(n+k)$，输出 $y(n+k)$，其中 $y(n)$ 是 $x(n)$ 的输出结果，则滤波器是移不变的。

具有上述所有特征的滤波器称为线性时不变（Linear Time Invariant，LTI）滤波器。由于具有这些特性，因此这类滤波器可以级联或并联，分别如图 2-14a 和图 2-14b 所示。

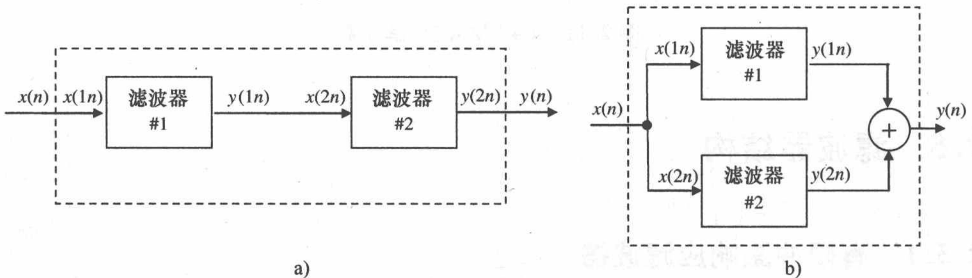


图 2-14　FIR 滤波器设置

a）级联 FIR 滤波器　b）并联 FIR 滤波器

设计数字滤波器的一种基本方法是从期望的频率响应开始的，对其进行逆滤波得到冲激响应，截断冲激响应，然后加窗处理来消除虚像（Bourke 1999，Williams 1986）。比如我们考虑典型的低通函数，如图 2-15 所示，转换到时域后，就得到经典的从零点开始的 sinc 函数。事实上，我们需要用有限数量的系

数来逼近这个无限长的滤波器，考虑到这种滤波器需要当前时刻之后的数据，将其作时移，从而时间负轴上不会有值。如果我们设计这种滤波器，并将其转换到频域，则会发现在截止频率和带通频率上会出现振荡或者纹波效应，并且通带到阻带区域之间的过渡带变化非常平缓。这种纹波通常称为吉布斯（Gibbs）现象，是由 Willard Gibbs 在 1899 年发现的。

通过增加滤波器系数或抽头的数量可以抑制纹波效应，从而更好地逼近滤波器，但增加了计算开销。加窗也可以抑制纹波效应，这实际上可以看成是在原始频谱图上加矩形窗，其他所熟知的窗函数还有 Von Hann 窗，汉明（Hamming）窗和 Kaiser 窗（Lynn 和 Fuerst 1994，Williams 1986），它们以不同的方式从不同程度上减小纹波和过度带。滤波器设计的最终结果是确定滤波器长度和系数值，以尽可能满足滤波器响应的要求。FIR 滤波器的实现相对容易理解，因为时域和频域之间有着直接联系。FIR 滤波器还有以下的优点：

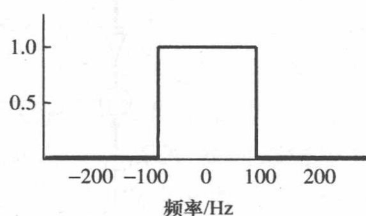


图 2-15　低通滤波器响应

1）线性相位，即输入信号的延迟不会导致相位失真；

2）固有的稳定性；

3）可以通过有限的算术运算实现；

4）对量化误差不敏感。

2.5.2　相关

相关是与数字通信有关的函数，可以通过式（2-13）计算。

$$y(n) = \sum_{i=-\infty}^{\infty} a_i x(n+i) \tag{2-13}$$

式（2-13）对于有限长序列的相关，即 $x(n)$，$n=0, 1, \cdots, N-1$，表达式可写为

$$y(n) = \sum_{i=0}^{N-1} a_i x(n+i) \tag{2-14}$$

式（2-14）与式（2-11）中的 FIR 滤波器表达式相似，只是数据流 x_n 的顺序正好相反。相关运算的结构也与 FIR 滤波器相似。

2.5.3　无限冲激响应滤波器

FIR 滤波器的主要缺点是需要很多抽头来实现频率响应的某些特性，如迅速截止，使得计算成本很高。无限冲激响应（Infinite Impulse Response，IIR）滤波

器通过使用以前的输出结果可以克服这个缺点，如式（2-15）所示。式（2-16）的传输函数表达式可以很好地解释，图 2-16 所示为 IIR 滤波器的结构。

$$y(n) = \sum_{i=0}^{N-1} a_i x(n-i) + \sum_{j=1}^{M-1} b_j y(n-1) \tag{2-15}$$

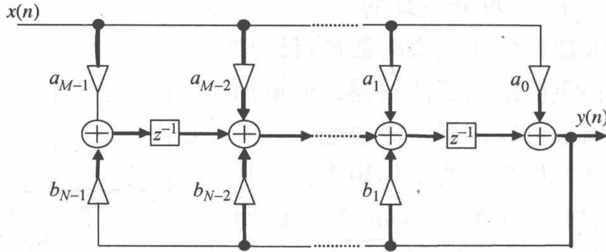


图 2-16 直接形式 IIR 滤波器

IIR 滤波器的设计过程与 FIR 滤波器不同，通常采用模拟滤波器的设计方法。通过模拟滤波器 s 平面和 z 域（Grant 等 1989）之间的变换，可以实现数字滤波器。人们提出了很多设计方法，比如冲激不变法（Williams 1986），z 变换匹配法和 Bilinear 变换（Grant 等 1989，Williams 1986）。设计的结果是得到一个传输函数表达式，包括式（2-17）中的零点和极点。主要的问题是要维持稳定性，只要能确保极点处于单位圆内就可实现。这些零点和极点的位置与滤波器的特性有直接关系。比如，如果单位圆上的极点没有零点来抵消，则这种滤波器将在某些频率上输出无穷大值（Meyer-Baese 2001）。

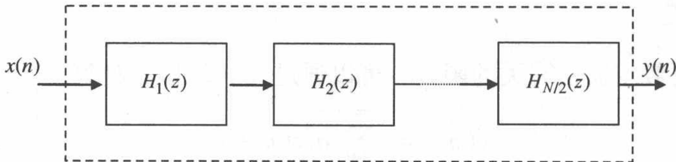


图 2-17 二阶 IIR 滤波器模块级联

$$H(z) = \frac{\sum_{i=0}^{N-1} a_i x_{n-i}}{1 - \sum_{j=1}^{M-1} b_j y_{n-j}} \tag{2-16}$$

$$H(z) = G\frac{(z-\xi_1)(z-\xi_2)\cdots(z-\xi_M)}{(z-p_1)(z-p_2)\cdots(z-p_N)} \tag{2-17}$$

由于存在图 2-16 中所示的反馈回路，这种结构对量化误差非常敏感，并且会随着滤波器阶数增长而增加。因此，滤波器通过式（2-18）的二阶 IIR 滤波器

的级联来实现，组成图2-17所示的结构，图中，每个$H_j(z)$模块都是一个二阶IIR滤波器，$j=1,2,\cdots,N/2$。

$$y(n)=a_0x(n)+a_1x(n-1)+a_2x(n-2)+b_1y(n-1)+b_2y(n-2) \quad (2\text{-}18)$$

2.5.4 波形数字滤波器

除了FIR滤波器和IIR滤波器，还有一类波形数字滤波器（Wave Digit Filter，WDF）的滤波器结构也广泛使用，它们对系数变化不太敏感。这个特点非常重要，因为在IIR滤波器中，系数变动决定了实现滤波器系数所能达到的精度，并且与滤波器结构所需要的动态范围有直接关系。从硬件角度看，内部字长的调整及滤波器性能非常重要，其必然会影响吞吐速率。WDF对内部结构的精度所导致的变化不太敏感，因此减少了系数变化所导致的响应失真程度。这对很多DSP应用都很重要，有几方面的原因：①它可以用较短字长的系数来满足滤波器的规格，从而使得硬件开销较小；②对系数敏感度低的结构产生的近似误差也较小，这里，近似误差由滤波器结构的有限算术精度所致（后面将讨论截短问题和字长导致的误差）。对于IIR滤波器，设计的出发点是利用模拟滤波器中低敏感度的特点来生成低敏感度的数字滤波器。

WDF代表了一类以经典模拟滤波器网络为基础建模的滤波器（Fettweis和Nossek 1982，Fettweis等 1986，Wanhammar 1999），模拟滤波器网络通常配置呈格型或阶梯型结构。对于工作在低频的电路，电路的尺寸比波长小，设计者可以将电路当作集总无源器件或集总有源器件之间的连接，且在电路中的任何一点都具有唯一的电压和电流值，依据是电路方面的相位变化可以忽略。针对这种结构有很多电路级设计优化技术可应用，比如Kirchoff定律。然而，对高频电路，上述假设不再成立，可能需要解麦克斯韦（Maxwell）方程。为此，设计者可以考虑一个事实，即他们是要解决某些特定位置的问题，比如终端电压和电流（Pozar 2005）。利用一些特殊类型的电路，比如具有相同电传播时间的传输线，可以将电路当作传输线器件，这种器件可以作为分布元件，其特性由长度、传播常数和特征阻抗决定。

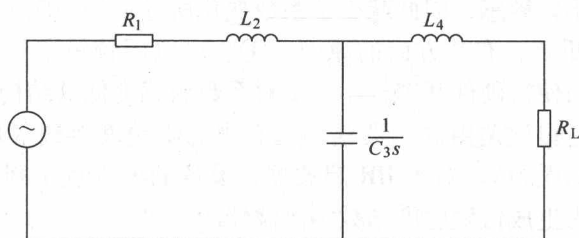
Fettweis等人（1986）深入地讨论了生成WDF的过程。主要的设计方法是利用传输线滤波器来产生滤波器，要将传输线滤波器与使用集总电路元件的经典滤波器联系起来，从而可以利用这些结构的特性，我们称之为参考滤波器。利用一个称为Richard变量的复频率变量ψ来映射参考结构，可以实现WDF与参考滤波器的一致性，该方法可以将参考结构有效地映射到ψ域。使用参考结构可以将其全部固有的无源性及无损特征传递到数字域，因此可以获得很好的滤波器性能，同时也降低了系数敏感性，从而可以使用较短的字长。Fettweis等人（1986）给出了最简单且最合适的ψ，如式（2-19）所示的z变量的双线性变换。

$$\psi = \frac{z-1}{z+1} = \tanh(\rho T/2) \tag{2-19}$$

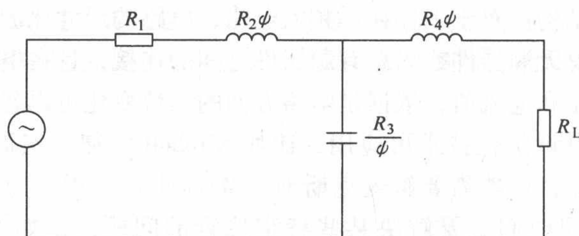
式中，ρ 表示实际的复频率。ψ 变量的一个特点是，实频率 ω 通过式（2-20）与实频率 ϕ 关联。

$$\phi = \tan(\omega T/2), \rho = \mathrm{j}\alpha, \psi = \mathrm{j}\phi \tag{2-20}$$

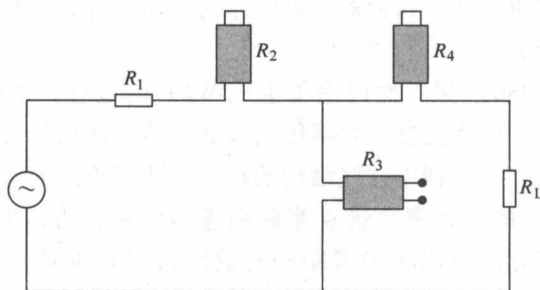
式（2-20）表明，参考域的实频率对应数字域的实频率。还有一个特点是，滤波器是因果的（Fettweis 等 1986）。图 2-18（Wanhammar 1999）所示为 WDF 滤波器设计的基本原理。图 2-18a 所示为一个集总元件滤波器，其中的多个无源器件 L_2s，$1/C_3s$ 和 L_4s 分别映射到图 2-18b 模拟滤波器中的 $R_2\psi$、R_3/ψ 及 $R_4\psi$。图 2-18c 表示利用式（2-19）映射等效传输线电路得到的 ψ 域滤波器。



a)



b)



c)

图 2-18　波形数字滤波器构造（Wanhammar 1999）

a）引用集总参数元件滤波器　b）域滤波器结构　c）产生的双端滤波器

WDF 组成模块

如图 2-18c 所示，基本 WDF 的构造由多个单端口、双端口和多端口元件组成。图 2-19 所示为双端口元件的基本描述。网络可以描述为入射波 A_1 和反射波 B_2，它们与端口电流 I_1 和 I_2、端口电压 V_1 和 V_2、端口电阻 R_1 和 R_2 有关（Fettweis 等 1986）。

$$A_1 \cong V_1 + R_1 I_1 \tag{2-21}$$

$$B_2 \cong V_2 + R_2 I_2 \tag{2-22}$$

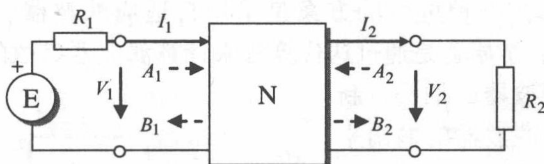


图 2-19 双端口适配器的基本描述

式（2-23）表示其传输函数 S_{21}。

$$S_{21} = KB_2/A_1 \tag{2-23}$$

式中，K 为

$$K = \sqrt{R_1/R_2} \tag{2-24}$$

Fettweis 等人（1986）在一篇文章中证明损失 α 与电路参数有关，即电感系数或电容系数、频率 ω，对于设计周到的滤波器，在通带内对幅度的敏感度较小，因此可以使用较短的系数字长。

参考滤波器基本结构由多个普通的双端口和三端口元件或适配器组成，如图 2-18c 所示。更详细的模块定义可参考 Fettweis 等人的著作（Fettweis 等 1986，Wanhammar 1999），主要是组成乘法器和加法器。

2.6 自适应滤波

第 2.2~2.5 节描述了广泛应用的一些变换和滤波算法。通常，它们可用于时频域的变换，如利用 FFT 或 DCT，或者用滤波算法来识别信号中的某些特征，比如 EEG 滤波。然而，也有很多应用其信号的有意义的区域未知，需要其他的滤波算法来完成，这就是我们所熟知的自适应算法。从高速实现的角度看，实现这类算法是一种挑战。本书专门有一章讨论自适应滤波器的硬件实现。

2.7 自适应滤波基础

滤波器的基本功能是从信号中去除不需要的部分。实现最优设计通常需要知

道有用信号中某些统计量（比如均值和相关函数）的先验知识。获得这些信息后，就可以设计最优滤波器，能够根据一些统计上的准则来减少无用信号。一种常用的方法是令滤波器的理想响应与实际响应之间的误差信号的均方值最小。最小化可得到一个针对平稳输入信号的唯一最优设计的代价函数，即维纳（Wiener）滤波器（Widrow 和 Hoff 1960）。然而，维纳滤波器只有当输入数据的统计特征与设计滤波器时所依赖的先验信息匹配时才是最优的，因此当输入信号的统计特性未知或者信号是非平稳的时候，就不再是最优设计。此时，需要时变滤波器来应对这种变化。一种可行的方案是采用自适应滤波器，可以形象地称为“自设计”滤波器，本质上是通过迭代算法来计算滤波器参数的更新值。用这些更新值来计算新滤波器的抽头，新的输入数据利用新滤波器的输出来构成下一组参数的更新值。如果输入的是平稳信号（Haykin 2001），则滤波器的算法经过几次迭代之后将按照设定的准则收敛到最佳方案。如果信号是非平稳的，则算法将跟踪输入信号的统计量变化，成功与否取决于算法本身的收敛速度和输入信号统计特性变化的快慢。

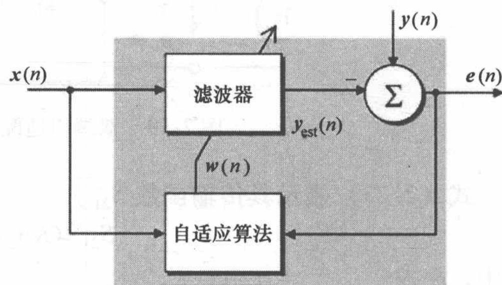


图 2-20 通用自适应滤波器系统结构

图 2-20 所示为自适应滤波器的通用结构。图中，信号 $\boldsymbol{x}(n)$ 同时输入 FIR 滤波器和自适应算法模块。自适应 FIR 滤波器的输出 $\boldsymbol{y}_{est}(n)$ 表示对期望信号 $\boldsymbol{y}(n)$ 的估计。误差信号 $\boldsymbol{e}(n)$ 表示 $\boldsymbol{y}(n)$ 与 $\boldsymbol{y}_{est}(n)$ 的差。自适应滤波器使用误差信号和输入信号 $\boldsymbol{x}(n)$ 来计算滤波器权重 $\boldsymbol{w}(n)$ 的更新（信息）。图中所绘的自适应滤波器结构适用于多种应用，2.7.1 节将讨论其中的部分应用。

2.7.1 自适应滤波器的应用

由于自适应滤波器能够工作在非平稳条件下，因此自适应滤波器已经成为那些输入信号统计特性未知或变化的 DSP 应用的重要部分。比如，信道均衡、回升抵消或自适应波束形成等。基本功能可归结为利用自适应滤波器完成一系列工作，即系统辨识、逆系统辨识、噪声消除和预测。对于每种应用，都是从基本的滤波器结构衍生出不同的形式，如图 2-20 所示。本节将结合应用实例来详细说明这些自适应滤波器的实现过程。

图 2-21 所示为自适应滤波器在系统辨识中的应用。本例中，自适应滤波器的目的是对未知系统 H 建模，H 的冲激响应用 $h(n)$ 表示，$n=0,1,2,3,\cdots,\infty$，$n<0$ 时 $h(n)$ 为 0。$\boldsymbol{x}(n)$ 同时输入到 H 和自适应滤波器。与前面的例子相似，先计

算误差信号 $e(n)$，然后用它来计算滤波器权重的更新（信息）$w(n)$，进而计算未知系统 $h(n)$ 的估计。在电信系统中，由于公共交换电话网络中的混合器器件阻抗不匹配会导致电话线中出现回声，因此，自适应系统可以先通过训练对未知回声路径建模，滤波器通过合成回声信号生成反向的回声路径，然后从原始接收信号中减去回声干扰信号。

图 2-22 所示为逆系统辨识。与图 2-21 的示例相似，自适应滤波器会尝试建模，但这里是对未知系统 H 的逆系统建模，从而可以抵消未知系统的效果。典型的应用是信道均衡（Drewes 等 1998），通过对信道干扰特征的逆效应建模，可以去除符号间干扰和传输信道中的噪声。

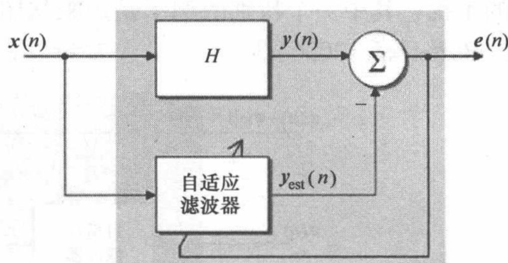


图 2-21 系统辨识

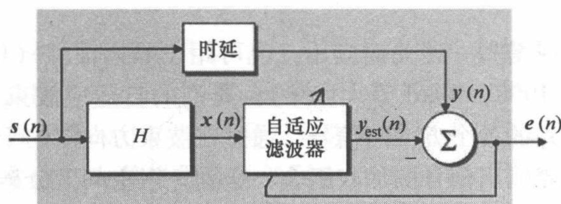


图 2-22 逆系统辨识

图 2-23 所示为基于自适应滤波器的预测系统。典型的应用是电话中的音频压缩。自适应差分脉冲编码调制（Adaptive Differential Pulse Code Modulation, ADPCM）算法会预测下一个音频样本，然后只对预测值和实际值的差进行编码和传输。通过这种方法，语音的数据速率可以降到 32kbit/s 的一半，同时还保持原来的质量。国际标准（国际电信联盟 International Telecommunication Union, ITU）给 ADPCM 定义的结构是一个基于 IIR 的两个极点、6 个零点的自适应预测器（ITU-T 1990）。

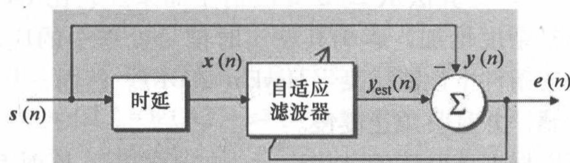


图 2-23 预测系统

图2-24所示为自适应噪声消除的结构，它与前面的示例稍有不同。图中，参考信号是数据 $s(n)$ 和噪声 $v(n)$ 的叠加信号。输入到自适应滤波器的噪声 $v'(n)$ 与噪声 $v(n)$ 强相关，但与理想信号 $s(n)$ 不相关。很多应用中使用了这种预测系统。比如可用于电话网络的回声抵消，也可用于免提电话中的声学回声消除。在医学应用中，噪声消除可用来去除心电图（Electrocadiogram，ECG）信号中的干扰，其中一个特别的例子是用来从胎儿的ECG曲线中去除母体的心跳信号（Baxter 等 2006）。

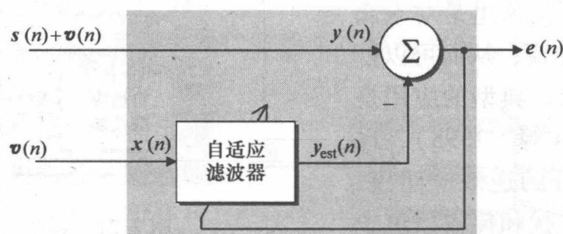


图2-24 噪声消除

自适应波束形成是另一种关键应用，也可用于噪声消除（Litva 和 Lo 1996，Moonen 和 Proudler 1998，Ward 等 1986）。典型的自适应波束形成器的功能是抑制除预期方向以外的各个方向的信号，通过在波束方向图的干扰方向引入零陷来实现。波束形成器输出信号的加权组合，送到一组空间上分离的天线，包含一根主天线和多根辅助天线。主信号来自主天线，具有很强的方向性。辅助信号含有干扰信号的样本，有可能淹没理想信号。滤波器通过去除任何与主输入信号有共同特点的信号（即与干扰噪声强相关的信号）来消除干扰。将来自辅助天线和主天线的数据输入自适应滤波器，然后计算权重，将这些权重用于延迟的输入数据来产生输出波束。第12章将详细讨论。

2.7.2 自适应算法

自适应滤波巧妙的地方在于计算更新的滤波器权重的算法。主要有两种算法，递归最小二乘（Recursive Least Square，RLS）算法和最小均方（Least Mean Square，LMS）算法。RLS算法从LMS算法衍生而来。它比LMS算法收敛速度快，但代价是计算复杂度增加，影响其在实时信号处理中的应用。LMS算法简单而有效，在合适的条件下性能优良（Haykin 2001）。然而，其缺点是对输入数据矩阵的条件数敏感，并且收敛速度慢。

滤波器系数可以根据滤波器的结构表示为抽头权重、反射系数或旋转参数，分别对应横向结构、格型结构或收缩阵列（Haykin 1986）。LMS和RLS算法可用于图2-13所示的横向滤波器的基本结构，先由一个线性合成器计算系统输入

$\boldsymbol{x}(n)$的加权和，然后将其从期望信号$\boldsymbol{y}(n)$中减掉，得到误差信号$\boldsymbol{e}(n)$，如式（2-25）所示。在图 2-20 中，$\boldsymbol{w}(n)$表示自适应算法的权重矢量。

$$\boldsymbol{e}(n) = \boldsymbol{y}(n) - \sum_{i=0}^{N-1} \boldsymbol{W}_i \boldsymbol{x}_{n-i} \tag{2-25}$$

目前没有确切的方法来确定针对某个特殊应用的最佳自适应算法。算法的选择归结于与算法有关的多种特征之间的平衡，比如

1）收敛速度，即自适应算法达到最优方案附近可接受的误差范围内的速度；

2）稳态误差，即最优方案的逼近；

3）跟踪输入数据统计量变化的能力；

4）计算复杂度；

5）处理病态输入数据的能力；

6）对实现所用的字长变化的敏感度。

2.7.3 LMS 算法

LMS 算法是一种随机梯度算法，它利用固定步长参数来控制横向滤波器抽头权重的更新，如图 2-20 所示（Widrow 和 Hoff 1960）。算法的目的是使$\boldsymbol{y}(n)$和$\boldsymbol{y}_{\text{est}}(n)$之间的均方误差最小。均方误差与未知的抽头权重之间的关系构成一个多维抛物面，通常称之为误差曲面（如图 2-25 所示的二抽头横向滤波器的误差曲面）（Haykin 2001）。曲面有唯一的最小点表示最佳维纳解的抽头权重（定义为 Wiener-Hopf 方程，详见 2.7.4 节）。然而，在非平稳条件下，误差曲面不停地变化，因此 LMS 算法需要能够跟踪曲面的底部。LMS 算法通过在每个离散时刻n选择合适的权重矢量$\boldsymbol{w}(n)$来达到代价函数$V(\boldsymbol{w}(n))$最小化的目的。策略是根据瞬时梯度值$\mathrm{d}V(\boldsymbol{w}(n))/\mathrm{d}\boldsymbol{w}(n)$来更新参数估计，即

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \mu \frac{\mathrm{d}V(\boldsymbol{w}(n))}{\mathrm{d}\boldsymbol{w}(n)} \tag{2-26}$$

式中，μ是一个正的步长，前面的负号是确保参数估计误差沿着误差曲面下降。

代价函数$V(\boldsymbol{w}(n))$的目的是使均方误差最小，可以得到式（2-27）所示的递归参数更新方程。

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \mu \boldsymbol{x}(n)(\boldsymbol{y}(n) - \boldsymbol{y}_{\text{est}}(n)) \tag{2-27}$$

用于更新式（2-26）抽头权重矢量的递归表达式可写为

$$\boldsymbol{w}(n+1) = \boldsymbol{w}(n) - \mu \boldsymbol{x}(n)(\boldsymbol{y}(n) - \boldsymbol{x}^{\mathrm{T}}(n)\boldsymbol{w}(n)) \tag{2-28}$$

式（2-28）可进一步分解成式（2-29）所示的滤波器输出，式（2-30）所示的估计误差和式（2-31）所示的抽头权重自适应更新过程。

$$\boldsymbol{y}_{\text{est}}(n) = \boldsymbol{w}^{\mathrm{T}}(n)\boldsymbol{x}(n) \tag{2-29}$$

$$e(n) = y(n) - y_{est}(n) \tag{2-30}$$

$$w(n+1) = w(n) + \mu x(n)e(n) \tag{2-31}$$

对于 N 抽头的权重矢量，LMS 算法每次迭代过程只需要 $2N+1$ 次乘法和 $2N$ 次加法。因此，它有相对简单的结构，硬件资源和权重数成正比。

2.7.4 RLS 算法

相对 LMS 算法而言，RLS 是一种计算复杂的算法，它由最小二次方（Least Square，LS）算法推导而来。LS 算法的代价函数 $J(n)$ 的目标是令平方误差和最小，如式（2-32）所示。

$$J(n) = \sum_{i=0}^{N-1} | e(n-i) |^2 \tag{2-32}$$

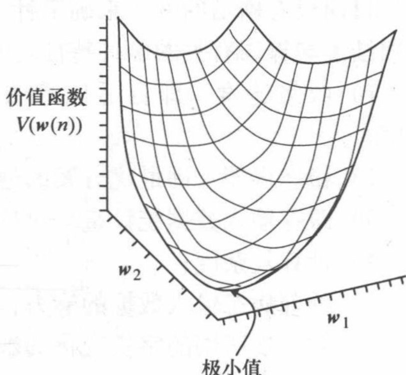


图 2-25 二抽头横向滤波器误差曲面（Haykin 2001）

将式（2-25）代入式（2-32），得到

$$J(n) = \sum_{i=0}^{N-1} \left| y(n) - \sum_{i=0}^{N-1} w_k x(n-i) \right|^2 \tag{2-33}$$

将离散时域的信号序列转换为矩阵－矢量形式可以简化方程表达式。考虑样本数为 N 的数据值，式（2-25）可以写成

$$e(n) = \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_N \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix} - \begin{bmatrix} \underline{x}_1^{T} \\ \underline{x}_2^{T} \\ \vdots \\ \underline{x}_N^{T} \end{bmatrix} \begin{bmatrix} W_1 \\ W_2 \\ \vdots \\ W_N \end{bmatrix} \tag{2-34}$$

式（2-34）用矢量表示如下：

$$e(n) = y(n) - X(n)w(n) \tag{2-35}$$

代价函数 $J(n)$ 可以表示成如下的矩阵形式：

$$J(n) = e(n)^{T}e(n) = (y(n) - X(n)w(n))^{T}(y(n) - X(n)w(n)) \tag{2-36}$$

将乘积项展开，经过简化后可得

$$J(n) = y^{T}(n) - 2y^{T}(n)X(n)w(n) + w^{T}(n)X^{T}(n)X(n)w(n) \tag{2-37}$$

式中，$X^{T}(n)$ 表示 $X(n)$ 的转置，$y^{T}(n)$ 表示 $y(n)$ 的转置。为了找到最佳权重矢量，对式（2-37）求关于 $w(n)$ 的微分，令微分等于零来求解权重矢量。从式（2-37）可以推导得出最小平方权重矢量的估计，即 $w_{LS}(n)$，用矩阵形式表示如下：

$$w_{LS}(n) = (X^{T}(n)X(n))^{-1}X^{T}y(n) \tag{2-38}$$

上面的表达式可以用 Wiener－Hopf 标准式表示［式（2-39）～式

(2-41)]。

$$w_{LS}(n)=\boldsymbol{\phi}(n)^{-1}\boldsymbol{\theta}(n) \tag{2-39}$$

$$\boldsymbol{\phi}(n)=\boldsymbol{X}^{T}(n)\boldsymbol{X}(n) \tag{2-40}$$

$$\boldsymbol{\theta}(n)=\boldsymbol{X}^{T}(n)\boldsymbol{y}(n) \tag{2-41}$$

式中，$\boldsymbol{\phi}(n)$表示输入数据$\boldsymbol{X}(n)$的相关矩阵，$\boldsymbol{\theta}(n)$表示输入数据$\boldsymbol{X}(n)$和期望信号矢量$\boldsymbol{y}(n)$的互相关矢量。假设观察次数大于权重数，因为方程数大于未知数的个数，所以上面的方程有解。

上面的LS解是针对样本数据输入块来执行的，该解可以用RLS算法迭代实现，此时，LS权重会对每组新样本输入都更新。如果持续执行，则LS算法可以有效地运行在无限大的数据窗上，因此适合于稳态系统。非平稳条件应用的LS解可以包含一个加权因子。该因子可以赋予最新输入的数据更重要的权重，从而有效地构建了一个数据的移动窗用来获得LS解。引入遗忘因子β后，式(2-36)中的LS代价函数写为

$$\boldsymbol{J}(n)=\sum_{i=0}^{N-1}\beta(n-i)e^{2}(i) \tag{2-42}$$

式中，$\beta(n-i)$定义为$0<\beta(n-i)\leqslant 1$，$i=1,2,\cdots,N$。遗忘因子的一种形式是指数遗忘因子。

$$\beta(n-i)=\lambda^{n-i} \tag{2-43}$$

式中，$i=1$，2，…，N，λ为接近但小于1的正常数。由于该值决定了将使用的数据窗长度并且会影响自适应滤波器的性能，因此尤为重要。$(1-\lambda)$的倒数可以度量算法的记忆长度。通常的规则是，系统记忆长度越长，收敛速度越快，稳态误差越小。然而，系统统计特性的变化速度会限制窗口长度。将遗忘因子应用到Wiener-Hopf标准方程［式(2-39)～式(2-41)］中，相关矩阵和互相关矩阵可写成

$$\boldsymbol{\phi}(n)=\sum_{i=0}^{n}\lambda^{n-1}\underline{\boldsymbol{x}}(i)\underline{\boldsymbol{x}}^{T}(i) \tag{2-44}$$

$$\boldsymbol{\theta}(n)=\sum_{i=0}^{n}\lambda^{n-1}\underline{\boldsymbol{x}}(i)\underline{\boldsymbol{y}}(i) \tag{2-45}$$

因此，迭代表达式可写为

$$\boldsymbol{\phi}(n)=\left[\sum_{i=1}^{n-1}\lambda^{n-i-1}\underline{\boldsymbol{x}}(i)\underline{\boldsymbol{x}}^{T}(i)\right]+\underline{\boldsymbol{x}}(n)\underline{\boldsymbol{x}}^{T}(n) \tag{2-46}$$

或者更简洁地表示为

$$\boldsymbol{\phi}(n)=\lambda\boldsymbol{\phi}(n-1)+\underline{\boldsymbol{x}}(n)\underline{\boldsymbol{x}}^{T}(n) \tag{2-47}$$

相似地，$\boldsymbol{\theta}(n)$可以表示为

$$\boldsymbol{\theta}(n)=\lambda\boldsymbol{\theta}(n-1)+\underline{\boldsymbol{x}}(n)\underline{\boldsymbol{y}}(n) \tag{2-48}$$

从求解 Wiener – Hopf 标准方程得到 LS 权重矢量需要计算相关矩阵的逆，例如式（2-49）所示的矩阵矢量的表达式。

$$\begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \underbrace{\left[\begin{bmatrix} \boldsymbol{X}_{11}\boldsymbol{X}_{12}\boldsymbol{X}_{13} \\ \boldsymbol{X}_{21}\boldsymbol{X}_{22}\boldsymbol{X}_{23} \\ \boldsymbol{X}_{31}\boldsymbol{X}_{32}\boldsymbol{X}_{33} \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} \boldsymbol{X}_{11}\boldsymbol{X}_{12}\boldsymbol{X}_{13} \\ \boldsymbol{X}_{21}\boldsymbol{X}_{22}\boldsymbol{X}_{23} \\ \boldsymbol{X}_{31}\boldsymbol{X}_{32}\boldsymbol{X}_{33} \end{bmatrix} \right]^{-1}}_{\text{相关矩阵}} \cdot \underbrace{\left[\begin{bmatrix} \boldsymbol{X}_{11}\boldsymbol{X}_{12}\boldsymbol{X}_{13} \\ \boldsymbol{X}_{21}\boldsymbol{X}_{22}\boldsymbol{X}_{23} \\ \boldsymbol{X}_{31}\boldsymbol{X}_{32}\boldsymbol{X}_{33} \end{bmatrix}^{\mathrm{T}} \begin{bmatrix} y_{11} \\ y_{12} \\ y_{13} \end{bmatrix} \right]}_{\text{互相关矩阵}} \tag{2-49}$$

逆矩阵的存在给实现带来了难题，包括数值稳定性和计算复杂度。比如，如果相关矩阵是奇异的，则算法将面临数值稳定性问题。同时，每次迭代过程中计算矩阵的逆复杂度为 N^3 的量级，而 LMS 算法复杂度仅为 N 的量级。有两种特殊的方法来迭代地求 LS 解，它们不需要直接计算矩阵的逆，从而将复杂度降到 N^2 量级。第一种方法称为标准 RLS 算法，利用矩阵求逆引理来迭代更新权重。一种普遍使用的方法是首先对输入数据进行一组正交旋转，即 Givens 旋转（Givens 1958），将正方形数据矩阵变换到等效的上三角矩阵（Gentleman 和 Kung 1981），然后通过回代来计算权重。这就是人们熟知的 QR 分解（利用一种正交旋转方法来实现，比如 Householder 变换或者 Givens 旋转），它已经成为数值稳定性和鲁棒性 RLS 算法的基础（cioffi 1990，Cioffi 和 Kailath 1984，Dohler 1991，Hsieh 等 1993，Liu 等 1990，1992，McWhirter 1983，McWhirter 等 1995，Rader 和 Steinhardt 1986，Walke 1997）。目前，有多种快速 RLS 算法，它们通过控制系统中的冗余来将复杂度减少到 N。

2.8 总结

本章简要地介绍了 DSP 的基础知识及一些通用的 DSP 算法，包括 DCT、FFT 和 DWT 等变换，基本滤波器结构如 FIR 和 IIR 滤波器，以及 LMS 和 RLS 等自适应滤波器。当然，类似的信号处理算法有很多，本章只重点介绍了其中的一部分，本书后续章节设计实例时还将用到。此外，本书专门有一章详细介绍用来构建复杂核的 RLS 滤波器结构。

参考文献

Baxter P, Spence G and McWhirter J (2006) Blind signal separation on real data: tracking and implementing *Proc. ICA-2006*, pp. 327–334, Charleston, USA.

Bourke P (1999) Fourier method of designing digital filters. Web publication downloadable from http://local.wasp.uwa.edu.au/ pbourke/other/filter/.

Cioffi J (1990) The fast householder filters rls adaptive algorithm RLS adaptive filter. *IEEE Proc. Int. Conf. on Acoustics, Speech and Signal Processing*, pp. 1619–1621.

Cioffi JM and Kailath T (1984) Fast recursive-least-square, transversal filters for adaptive filtering. *IEEE Trans. Acoustics, Speech, Signal Processing* ASSP-32(2), 998–1005.

Daubechies I (1992) Ten lectures on wavelets. *CBMS-NSF Regional Conference Series in Applied Mathematics*.

Dohler R (1991) Squared Givens' rotations. *IMA J. of Numerical Analysis* II, 1–5.

Drewes C, Hasholzner R and Hammerschmidt JS (1998) On implementation of adaptive equalizers for wireless atm with an extended QR-decomposition-based RLS-algorithm. *Proc. Int. Conf. on Acoustics, Speech and Signal Processing*, pp. 3445–3448.

Fettweis A and Nossek J (1982) On adaptors for wave digital filter. *IEEE Trans. Circuits and Systems* CAS-29(12), 797–806.

Fettweis A, Gazsi L and Meerkotter K (1986) Wave digital filter: Theory and practice. *Proc. the IEEE* 74(2), 270–329.

Gentleman WM and Kung HT (1981) Matrix triangularisation by systolic array. *Proc. SPIE (Real-Time Signal Processing IV)* 298, 298–303.

Givens W (1958) Computation of plane unitary rotations transforming a general matrix to triangular form. *J. Soc. Ind. Appl. Math* 6, 26–50.

Grant P, Cowan C, Mulgrew B and Dripps JH (1989) *Analogue and Digital Signal Processing and Coding*. Chartwell-Bratt Inc., Sweden.

Haykin S (2001) *Adaptive Filter Theory*. Prentice Hall, Englewood Cliffs, NJ.

Hsieh SF, Liu KJR and Yao K (1993) A unified approach for QRD-based recursive least-squares estimation without square roots. *IEEE Trans. Signal Processing* 41(3), 1405–1409.

ITU-T (1990) Recommendation g.726,40, 32, 24, 16 kbit/s adaptive differential pulse code modulation (ADPCM).

Jewett D (1970) Volume conducted potentials in response to auditory stimuli as detected by averaging in the cat. *Electroencephalography and Clinical Neurophysiology (Suppl.)* 28, 609–618.

Litva J and Lo TKY (1996) *Digital Beamforming in Wireless Communications*. Artec House, Norwood, MA.

Liu KJR, Hsieh S and Yao K (1990) Recursive LS filtering using block householder transformations. *Proc. Int. Conf. on Acoustics, Speech and Signal Processing*, pp. 1631–1634.

Liu KJR, Hsieh S and Yao K (1992) Systolic block householder transformations for rls algorithm with two-level pipelined implementation. *IEEE Trans. On Signal Processing* 40(4), 946–958.

Lynn P and Fuerst W (1994) *Introductory Digital Signal Processing with Computer Applications*. John Wiley & Sons, Chicester.

Mallat SG (1989) A theory for multiresolution signal decomposition: the wavelet representation. *IEEE Trans. Pattern Anal. Machine Intelligence* 11(7), 674–693.

McWhirter JG (1983) Recursive least squares minimisation using systolic array. *Proc. SPIE (Real-Time Signal Processing IV)* 431, 105–112.

McWhirter JG, Walke RL and Kadlec J (1995) Normalised givens rotations for recursive least squares processing. *Proc. VLSI Signal Processing, VIII*, pp. 323–332.

Meyer-Baese U (2001) *Digital Signal Processing with Field Programmable Gate Arrays*. Springer, Germany.

Moonen M and Proudler IK (1998) MDVR beamforming with inverse updating. *Proc. EUSIPCO*, pp. 193–196.

Nyquist H (2002) Certain topics in telegraph transmission theory. *American Telephone and Telegraph Co. New York, NY* 90(2), 280–305.

Omondi AR (1994) *Computer Arithmetic Systems*. Prentice Hall, New York.

Pozar DM (2005) *Microwave Engineering*. John Wiley & Sons, Inc., USA.

Rabiner L and Gold B (1975) *Theory and Application of Digital Signal Processing*. Prentice Hall, New York.

Rader CM and Steinhardt AO (1986) Hyperbolic householder transformations, definition and applications. *Proc. Int. Conf. on Acoustics, Speech and Signal Processing*, 11, 2511–2514.

Shannon CE (1949) Communications in the presence of noise. *Proc. IRE*, 37, 10–21.

Walke RL (1997) *High Sample Rate Givens Rotations for Recursive Least Squares*. PhD Thesis, University of Warwick.

Wanhammar1999 L (1999) *DSP Integrated Circuits*. Academic Press, San Diego.

Ward CR, Hargrave PJ and McWhirter JG (1986) A novel algorithm and architecture for adaptive digital beamforming. *IEEE Trans. On Antennas and Propagation* AP-34(3), 338–346.

Widrow B and Hoff CSJ (1960) Adaptive switching circuits. *IRE WESCON Conv. Rec.*, pp. 96–104.

Williams C (1986) *Designing digital filters*. Prentice Hall, New York.

第3章　算术运算基础

3.1　引言

算术算法的选择通常是DSP实现的一个重要方面，它不仅影响算法性能，还会影响系统性能指标，尤其是面积、速度和功耗。对于在处理器平台上的DSP实现而言，算术算法的选择变成了对平台的选择，通常是浮点实现或定点实现，接下来是对定点字长的选择。然而，对FPGA平台来说，算术算法的选择对性能代价的影响范围更大，贯穿整个设计过程；但公平地说，FPGA生产厂商（第5章将提到）对架构的决策将主导算术算法的选择。尽管如此，也值得我们考虑和理解一些算术表示细节。

DSP实现的一个关键前提是能否得到合适的基本处理元件，特别是加法器和乘法器。然而，一些DSP算法，特别是自适应滤波器同时还要求专门的器件来计算除法和二次方根。这些功能的实现及数字系统的选择会对硬件实现的质量带来重大影响。例如，众所周知，不同DSP应用领域（如图像处理、雷达和语音）对位切换有不同的要求，表现在变化次数和特殊位的切换上（Chandrakasan和Brodersen 1996）。更具体一点，语音输入的符号位会随着数据围绕零点波动而经常切换，但在图像处理中，输入通常是正值。此外，不同应用对较低有效位会有不同的切换行为（Chandrakasan和Brodersen 1996）。因此，有必要介绍一些计算机算术运算基础知识，特别是数值表示和某些通用算术运算，即加法器和乘法器实现方法的选择。但本书不打算触及太多细节，因为目前FPGA提供了专门的硬件来实现加法和乘法，而且对于很多应用，最少面积、最快速度和最低功耗的实现都将基于这些基本硬件单元。

尽管加法器和乘法器对DSP系统至关重要，但本书将重点介绍除法和二次方根运算，因为很多复杂DSP函数都用到这两种运算。本章还将简单比较用于实现这些运算的不同方法。动态范围是DSP的一个关键问题，涉及数据表示，即浮点和定点，本章将进行简单介绍，并回顾标记方法。

本章内容如下：3.2节将介绍一些计算机算术运算基础知识，包括多种形式的数字表示，其中将提及几种高级表示方法，比如有符号数字表示（SDNR）、逻辑数字系统（LNS）、余数系统（RNS）和坐标旋转数字计算方法（CORDIC）；3.3节将介绍定点和浮点表示方法；3.4节将简单介绍加法器和乘

法器的实现，并讨论一些 DSP 系统常用的复杂算术运算的实现，即除法和二次方根运算；最后将介绍定点和浮点的实现细节等关键问题，以及与算术运算和表示相关的其他问题的概括。

3.2 数字系统

我们从小就学习使用十进制表示来计数，但晶体管技术的发展意味着二进制计数方法是 DSP 系统中更自然的表示。本节将从传统数字系统的基本处理方法开始，解释带符号的级数表示和“1”的补码，然后重点介绍“2”的补码，因为它是当前最常用的表示方法。本节还将简单回顾其他一些在基于 FPGA 的 DSP 系统中使用的数字系统。

3.2.1 数字表示

假设 N 是 $n+1$ 位长的无符号数，可以用式（3-1）表示。

$$N = \sum_{i=0}^{n} x_i 2^i \tag{3-1}$$

式中，x_i 表示 N 的第 i 个二进制位，x_0 和 x_n 分别表示最高有效位（Most Significant Bit，MSB）和最低有效位（Least Significant Bit，LSB）。

1. 有符号量

在有符号技术系统中，$n-1$ 个较低有效位表示数量，最高有效位 x_n 表示符号。图 3-1a 用一个 4 位二进制字表示的数字轮形象地说明了这种表示方法。在符号数量标记法中，每个 4 位二进制字的量由 3 个较低有效位来决定，符号由最高有效位的符号位决定。然而，这种表示方法存在几个问题。首先，0 有两种表示，这是任何一种硬件系统都必须解决的问题，尤其是当 0 用于触发事件时，比如检查两个数是否相等时。其次，由于等式通常是逐位执行，因此这种方式增加了硬件复杂度。最后，减法这类运算将更加复杂，是因为如果不检查两个数字的字长并做相应的排列，则将无法校验结果值的符号位。

2. “1”的补码

在基于“1”的补码系统中，一个数的负值通过位翻转来实现，即由原始字中每位的“1”补码表示。式（3-2）给出了一个 n 位二进制字的转换方法，图 3-1b 以一个 4 位二进制字为例说明了这种表示方法。它仍然存在两个数表示 0 的问题，当用“1”的补码来做减法时需要对结果进行纠正（Omondi 1994）。

$$\overline{N} = (2^n - 1) - N \tag{3-2}$$

3. “2”的补码

在基于“2”的补码系统中，一个数的负值是在原码字逐位反转的基础上加

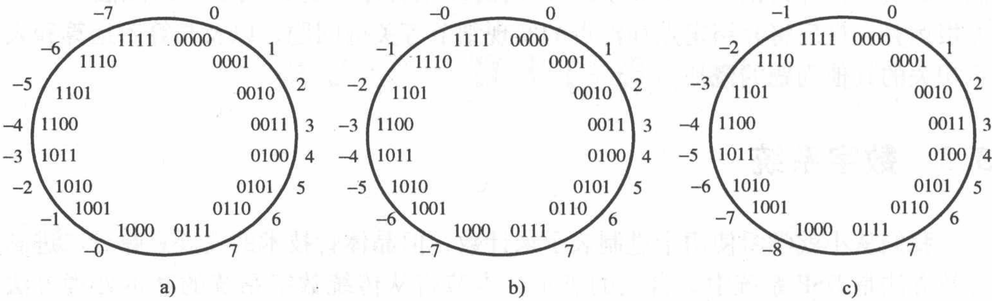


图 3-1　4 位二进制字的数字轮表示

a）符号和量级　b）“1”的补码　c）“2”的补码

1 得到的。式（3-3）给出了转换方法，图 3-1c 以 4 位二进制码字为例说明了这种表示方法。虽然这种方法看起来不如前面两种方法直观，但它有几个优势：①0 只有一种表示；②加减法可以简单地在硬件中实现，更重要的是减法，如果数字在有效表示范围内，则计算中的溢出可以忽略。因此，“2”的补码已经成为主流的数字系统表示方法。

$$\overline{N} = 2^n - N \tag{3-3}$$

4. 有符号数字值表示

有符号数字值表示（Signed Digit Number Representation，SDNR）最早由 Avizienies（1961）提出，用于算术运算中中断进位传播链。后来几位作者（Andrews 1986，Knowles 等　1989）提出一种有符号二进制数字表示（Signed Binary Number Representation，SBNR）方法，成功地应用到一些高速电路设计中，比如算术处理、数字滤波器和 Viterbi 译码函数等。同时允许使用正值和负值，意味着一个数可以有多种冗余表示。利用这个特点，Avizienis 演示了一个不需要进位传播的并行加法系统。当然，冗余表示必须转换回二进制，可以采用当时已有的多种技术来实现（Aklansky 1960）。

SBNR 表示促进了 SDNR 数字 x 二元集的发展，其中 $x \in (-1, 0, 1)$，严格地说是 $(\bar{1}, 0, 1)$，其中 $\bar{1}$ 表示 -1。这个二元集通常用 2 位来表示，即符号位 x_s 和数量位 x_m，见表 3-1。一种更有意思的配置是（+，-）方法，其中 SBNR 数字表示为 (x^+, x^-)，其中 $x = x^+ + (x^- - 1)$。或者可以理解为：当 $x^- = 0$ 时表示 -1，$x^- = 1$ 时表示 0；$x^+ = 0$ 时表示 0，$x^+ = 1$ 时表示 1。这种方法的主要优势在于它可以从传统加法器模块构建通用 SBNR 加法器，并且对构建高速乘法器非常关键。尽管 SDNR 表示从提高速度角度看具有性能优势，但与二进制之间的相互转换是一个需要考虑的问题，并且需要专用的电路（Sklansky 1960）。此外，内部数据的冗余表示给一些简单运算，比如比较运算带来问题。

表 3-1 SDNR 编码

SDNR 数字	SDNR 表达式			
	符号和量级		+/-	编码
x	x_s	x_m	x^+	x^-
0	0	0	0	1
1	0	1	1	0
$\bar{1}$	0	1	0	1
0 或 X	1	0	1	0

5. 其他表示方法

数字表示方法还有很多，比如逻辑数字表示（Logarithmic Number Representation，LNS）（Muller 2005）、余数表示（Residue Number Representation，RNS）（Soderstrand 等 1986）和坐标旋转数字计算（Coordinate Rotation Digital Computer，CORDIC）（Volder 1959，Walther 1971）。

在 LNS 中，数字 x 用定点数 i 表示，如式（3-4）所示。

$$i = \log_2 |x| \tag{3-4}$$

另外，还需要一个位来表示 x 的符号，特殊的情况是 $x=0$。LNS 的重要优点是线性域的乘法和除法可以简单地用对数（log）域的加法和减法来代替。然而，加法和减法却变得更复杂。参考文献（Collange 等 2006）介绍了 LNS 浮点库的发展，并讨论了它在某些算术函数和图像处理中的应用。

RNS 很适合处理数值很大的整数，因此多用在计算机算术系统和一些 DSP 应用中（后面会介绍），这些应用都存在对大数值整数计算的需求。在 RNS 中，一个整数将被转换为一组 N 元组的小整数，称为模数，记为（m_N，m_{N-1}，…，m_1）。整数 X 可以用 N 元组（x_N，x_{N-1}，…，x_1）表示如下，其中 x_i 为非负整数：

$$X = m_i \cdot q_i + x_i \tag{3-5}$$

式中，q_i 是满足的最大整数，x_i 是 X 模 m_i 的余数。RNS 主要的优点是加法、减法和乘法都不需要进位，因为其格式进行了转换。然而，对于除法、比较运算和符号判断等其他的算术运算，计算速度很慢，从而影响了 RNS 的大规模应用。因此，这种表示方法主要用在需要很多乘法和加法的 DSP 应用上，比如 FIR 滤波和 FFT 及 DCT 等变换（Soderstrand 等 1986）。

CORDIC 最早由 Volder（1959）提出，仅使用移位和加法操作就可实现旋转。因此很适合计算三角函数，比如正弦和余弦函数，另外还适合乘除运算，Walther（1971）将其应用到双曲函数、对数、指数和二次方根运算上，并首次提出适用于线性坐标、圆坐标和双曲坐标的通用算法。CORDIC 实现降低了旋转操作的计算量（Takagi 等 1991），它已经用到一些 DSP 应用中，尤其是矩阵三

角化（Ercegova 和 Lang 1990）和 RLS 自适应滤波（Ma 等 1997）的实现，后者也需要旋转操作。

上述的表示方法针对特定的实现，在某些应用上可以获得可观的性能增益，但应用范围受限。更重要的是，我们认为，它们相对现有方法所获得的性能增益并不一定值得推广使用。大多数 FPGA 架构都有基于传统算术算法的专用硬件，并且倾向于传统基于“2”的补码的处理。因此，本书的描述和示例都限定在“2”的补码。

3.3 定点和浮点

二进制数字系统中，一种广泛使用来表示和存储数值的格式是定点格式。定点算术运算中，整数 x 用一串位 x_{m+n-1}，x_{m+n-2}，…，x_0表示，其中 x_{m+n-1}，x_{m+n-2}，…，x_n表示数的整数部分，x_{n-1}，x_{n-2}，…，x_0表示数的分数部分。这是用户对数字系统的理解，在 DSP 系统中，用户通常将输入数据 $x(n)$ 和输出数据 $y(n)$ 用整数值表示，而系数值用分数表示，从而可以维持中间计算过程中的最佳动态范围。

选择定点表示方法的关键问题是在计算过程中充分利用动态范围。缩放可用于应对最坏情况，但通常会使动态范围变小。通过调整的方式来充分利用动态范围意味着某些情况下会发生溢出，需要增加额外的电路来处理。这个问题在基于“2”的补码表示方法中尤其突出，因为溢出所得到的值与原来的值符号完全不同。虽然可以通过饱和电路来保持最坏情况时的负溢出或正溢出，但其非线性区域对性能有影响，需要深入考虑。

上面提到的问题通常会在高层建模阶段考虑进去，常利用 MATLAB® 或者 LabVIEW 工具来分析。借用这些工具可以开发浮点表示的高层模型，然后转换成定点实现，任何溢出问题都在这时处理。处理方案可能会涉及 FPGA 实现，因为额外的电路可能导致时序问题。看上去好像麻烦比好处多，但定点实现对 FPGA 实现意义重大（对某些 DSP 微处理器实现也是一样），因为字长能直接转换为芯片面积。另外，现在有很多优化算法，使得定点表示很受欢迎；本章后面将继续讨论。

3.3.1 浮点表示

浮点表示为实数提供了一种更丰富的表示方法，越来越多地应用到科学计算应用中，同时也逐渐推广到 DSP 应用上。浮点表示的目的是用符号、指数和尾数或分数来表示一个实数，如图 3-2 所示。最常用的浮点格式是 IEEE 标准的二进制浮点运算（IEEE 754），定义了 4 种格式：

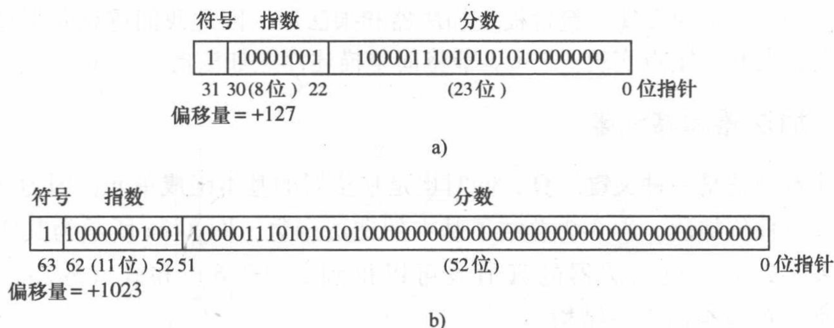


图 3-2　浮点表示
a）单精度　b）双精度

1）单精度（32 位）；

2）双精度（64 位）；

3）扩展单精度；

4）扩展双精度。

单精度 32 位表示方法中，用 1 位表示符号（S），用 8 位表示指数（Exp），剩下 23 位表示尾数（M），如图 3-2a 所示。数值 N 可以用 $2^{\text{Exp}127}\times \text{M}$ 来表示，由于指数是无符号的，因此单精度扩展表示的范围接近 $\pm 10^{38.53}$。双精度是简单地扩展到 64 位，表示的范围为 $\pm 10^{308.25}$。

下面的例子说明了如何将一个实数 -1082.5674 转换为 IEEE 754 表示的浮点数。

首先，因为是负数，可以确定 S=1；

将数（1082）用连除法转换到二进制（Omondi 1994），得到 10000111010；

小数部分（0.65625）用同样的方法转换为二进制，得到 10101；

两部分合起来得到 10000111010.10101；

基数小数点左移至只剩下一个整数位，得到 $1.000011101010101\times 2^{10}$；

补充 0 得到 23 位的尾数，即 10000111010101010000000；

指数是 10，32 位 IEEE 754 格式的偏置是 127，得到 137，用二进制表示为 10001001。

将这些数组合在一起得到如图 3-2a 所示单精度表示的二进制形式。采用同样的方法可得到图 3-2b 所示的双精度表示，主要的差别在于指数上的偏置及尾数后面填充的 0 的个数。

3.4　算术运算

本节讨论各种算术运算函数的实现，包括加法、乘法、除法和二次方根。本

书主要讨论 FPGA 的实现，包含板级加法器和乘法器，因此我们重点介绍这些构件的使用，尤其是定点实现。3.4.1 节将简单描述浮点加法器。

3.4.1 加法器和减法器

加法本身就是一种关键运算，同时也是乘法器的基本组成单元，因为乘法实际上是一组移位加法。表 3-2 总结了基本的加法函数，图 3-3a 所示为其实现过程。由表3-2 中 1 位加法器的真值表可以得到式（3-6）和式（3-7），以及图 3-3a所示的逻辑门实现结构。

$$S_i = A_i \oplus B_i \oplus C_{i-1} \tag{3-6}$$

$$C_i = A_i \cdot B_i + A_i \cdot C_{i-1} + B_i \cdot C_{i-1} \tag{3-7}$$

表 3-2 1 位加法器真值表

	输入			输出	
	A	B	C_i	S_o	C_o
$C_o = A + B$	0	0	0	0	0
	0	0	1	1	0
	0	1	0	1	0
$C_o = C_i$	0	1	1	0	1
	1	0	0	1	0
	1	0	1	0	1
$C_o = A + B$	1	1	0	0	1
	1	1	1	1	1

真值表还可以用另一种方式来解释：当 $A_i = B_i$ 时，$C_i = B_i$ 且 $S_i = C_{i-1}$；当 $A_i = \overline{B}_i$ 时，$C_i = C_{i-1}$且 $S_i = \overline{C}_{i-1}$。其中隐含了一种用来生成进位的乘法器，通过巧妙地利用 $A_i \oplus B_i$（计算和值 S_i 时的中间结果），只需要增加很小的开销。这是 FPGA 生产厂商喜欢用的结构，即如图 3-3b 所示的加法单元部分。通过提供专用的异或（Exclusive OR，EXOR）和多路选择（Multiplexer，MUX）逻辑，可以用 LUT 构建加法单元，并额外生成 EXOR 函数。

因为节省几百皮秒的运算时间对性能有很大的影响，所以在计算机算术运算领域有大量关于加法器结构的研究成果。人们开发了多种加法器结构，比如逐位进位加法器、超前进位加法器、进位保留加法器、跳跃进位加法器、条件求和等，这里就不一一列举了（Omondi 1994）。图 3-4 所示为一种逐位进位加法器，它是一种 4 位加法器的实现，每一个单元的逻辑表示都采用式（3-6）和式（3-7）。

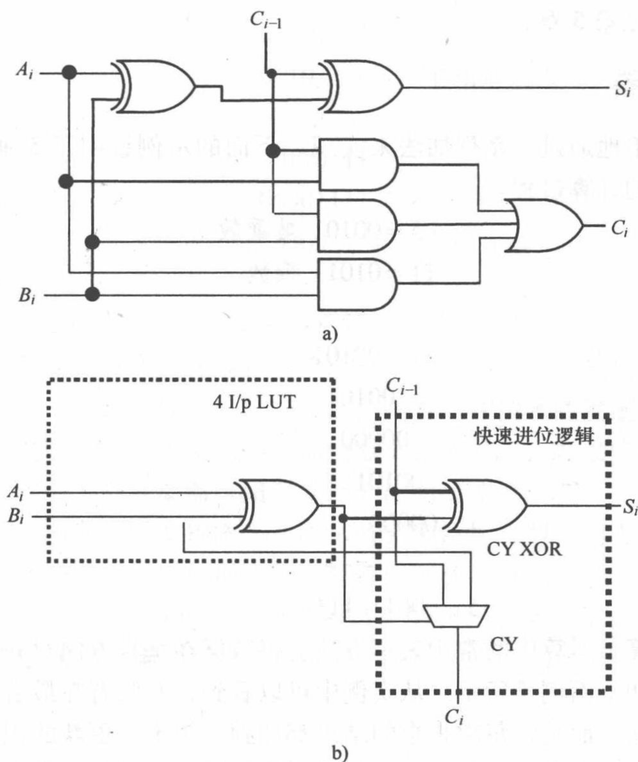


图 3-3　1 位加法器结构

a）传统的　b）基于多路复用的

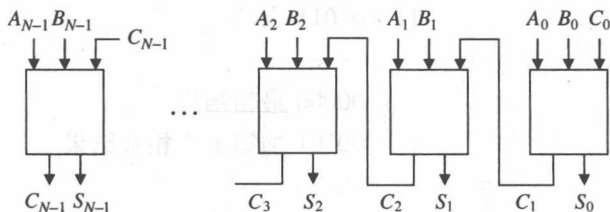


图 3-4　N 位加法器结构

不同加法器结构的变体很大程度上要权衡门电路的复杂度和系统规则性，很多技术都因为结构不够规则而终止。20 世纪 70 年代到 80 年代中，大量研究的目标是开发高速的结构，其中晶体管切换速度是主要特征。然而，如本书原书序中所分析的，互连才是最关键的，它在一定程度上削弱了采用先进加法器结构所带来的影响。另一个在 FPGA 中需要考虑的重要因素是加法器字长根据应用的需求进行缩放的能力，这种情况下，线性缩放会带来性能下降。

因此，并行加法器广泛应用在 FPGA 中，并且很多 FPGA 结构会为其提供专

门的资源（详见第 5 章）。

3.4.2 乘法器

乘法可简单地通过一系列加法来实现。下面的示例说明了 5 乘以 11 在二进制计数方法中的计算过程。

```
 5 =00101 被乘数
11 =01011 乘数
     ——
     00101
    00101
   00000
  00101
 00000
     ——
55 =000110111
```

在计算机算术运算中的常用表示方法会将数据在垂直方向对齐，并且向右移而不是左移，如下面例子所示。从实例中可以看到，并没有在最后才将每次相乘的结果一起相加，而是将每次相乘的结果都加到一个不停更新的积上，成为部分积。这意味着，计算过程中的每一步都等同于用与函数或门电路来实现每次的乘法，用加法器计算部分积。

```
 5 =00101 被乘数
11 =01011 乘数
   ——
   00000 最初结果
   00101 加第 1 个相乘结果
   ——
   00101
   000101 右移
   00101 加第 2 个相乘结果
   ——
   001111
   0001111 右移
   00000 加第 3 个相乘结果
   ——
   0001111
```

00001111 右移
00101 加第 4 个相乘结果
——
00110111
00001111 右移
00000 加第 5 个相乘结果
——
55 = 000110111

计算过程中的重复特点意味着每个阶段的顺序处理单元，包括一组与门来生成乘积项，一个加法器来生成逐阶段的加法。随着芯片技术的发展，并行乘法器已经成为主流，这就是说可以使用多个加法器电路。然而，如果使用图 3-4 所示的加法器单元，则会导致乘法器电路非常慢。应用快速加法器可以提高速度，但会增加硬件成本。因此，需要用图 3-5 所示的无进位加法器结构实现如图 3-6 所示的无进位阵列乘法器。无进位加法器计算速度和独立单元一样快，只有 3 个门延迟，参见图 3-3。这种结构可以快速得到最终的和，以及进位；然后用一个称为 CPA 的快速加法器来得到最终的和。

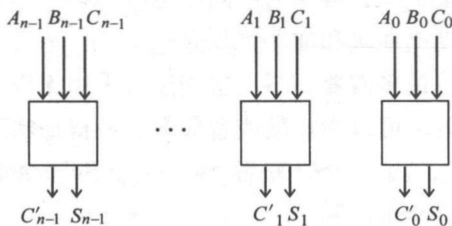


图 3-5　无进位加法器

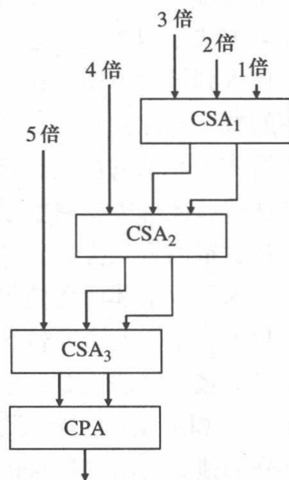


图 3-6　无进位阵列乘法器

每个加法阶段的速度都降到 2 个或 3 个门延迟，乘法器的速度则由总的阶段数决定。设字长为 m，阶段数量则为 $m-2$。Wallace 树状乘法器克服了这种局限，使得加法过程可以并行执行，如图 3-7 所示。无进位加法器将 3 个字压缩到两个字，意味着如果输入字长为 n，则在每个阶段后，n 个字变成 $3k+1$ 个，其中 $0 \leqslant k \leqslant 2$。因此，只需经过 $\log_{1.5} n$ 个阶段后得到最终的和与进位值，而无进位

阵列乘法器需要$n-1$个阶段。

3.4.3 除法

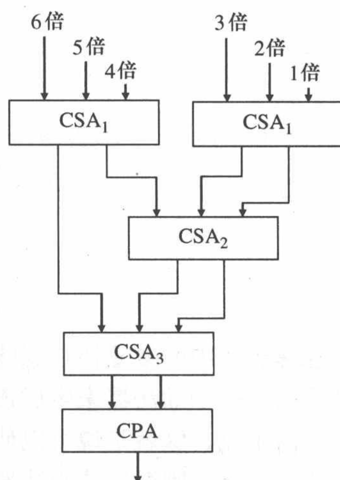


图 3-7 Wallace 树状乘法器

除法可能被认为是乘法的逆过程，但有几点区别使得其实现要复杂得多。完成除法有很多种方法，比如递推除法和函数迭代除法。从 20 世纪 50 年代开始，除法和二次方根的算法就已经成为计算机算术运算领域的重要研究课题。这些算法可以分成两类，即逐数字计算的方法和逼近方法。逐数字计算的方法也称为直接方法，与人们用纸笔计算商和二次方根的方法相似。其结果是一个数字一个数字计算，最高有效数字先计算。逼近方法包括牛顿 - 拉普森算法和泰勒序列展开算法，需要不停地更新近似数使其更加精确。

1. 递推除法

数字递推算法是广泛使用的减去法，在每次迭代中计算一个数字上的商。与手工计算方法相似的是，它们都按照从最高有效位到最低有效位的顺序计算。中间余数放入到被除数中，在每次迭代中，选择商数中与中间余数对应的数字。将商数中的数字与除数相乘，并将其从中间余数中减去。如果相减结果为负，则递推除法器将中间余数恢复到前一个值，即相减（比较）的结果决定算法下一次的除法迭代，此时需要从数字集中选出商数位。因此，商数位的选择要在每次迭代中通过试错的方法完成。这一点和乘法不同，乘法的中间积可以并行产生，在最后一起求和。上述问题使得除法的实现比乘法和加法更复杂。

这种方法中，两个n位的数相除可能最多需要$2n+1$次加法。采用无恢复递推算法可以减少加法次数，中间余数的数字可以为正数或者负数，使得加法次数减为n。最常用的递推除法是 SRT 算法，由三个研究者 Sweeney，Roberston 和 Tocher 分别独立提出（Robertson 1958，Tocher 1958）。

虽然递推算法可以通过简单的迭代实现，并且设计也不复杂，但延迟较大，并线性地收敛到商。每次迭代过程中退出计算的位数依赖于所使用的算术运算的基数。较大的基数可以减少迭代次数，但会增加每次迭代所需的时间。这是因为选择商数位的复杂程度随基数增加而呈指数增长，通常需要用 LUT。考虑基数和复杂度之间取折中，基数常取 2 或 4。

2. 函数迭代除法

前一节提到的数字递推算法在每次迭代中都有固定数量的位退出运算，算法

只使用移位和加法操作。与之不同的是，函数迭代算法采用乘法作为其基本操作，并且每次迭代过程得到的正确位的数量至少是数字递推算法的 2 倍（Flynn 1970，Ito 等 1995，Obermann 和 Flynn 1997，Oklobdzija 和 Ercegovac 1982）。因为每次迭代过程有 3 次乘法，这是一个重要的因素。然而，它具有二次收敛的优点，如图 3-8 所示，只需 6 次迭代就可得到 53 位的商。

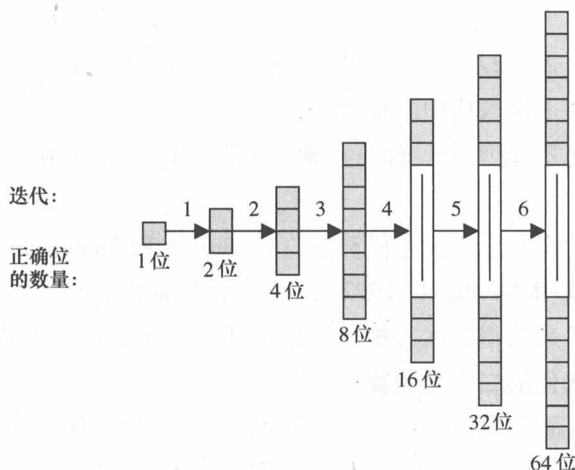


图 3-8 二次收敛

3.4.4 二次方根

计算二次方根的方法与除法相似，它们大致可分为两类，数字递推方法和基于逼近技术的方法。下面将简单介绍这两类算法。

1. 数字递推二次方根

数字递推算法既可以是恢复技术也可以是无恢复技术，两种算法都从最高有效位开始执行。算法是一种减去法，每次迭代结束后，如果出现负值，则将结果位置为 0，然后恢复原来的余数作为新的余数。如果数字为正值，则结果位置 1，并使用新的余数。无恢复算法可以保留负值，在下一次迭代中会做一次加法操作补偿回来。总体上来说，二次方根和除法算法非常相似，正因为如此，人们设计了很多脉动阵列的实现方法可同时用于这两种算术运算（Ercegovac 和 Lan 1991，Heron 和 Woods 1999）。

由于依赖于迭代次数及每行传递的进位，因此前面提到的算法性能上都有局限性。每个阶段都需要计算全部的值，才能正确地比较并做出决策。SRT 算法是一类无恢复的逐数字运算算法，其中的数字既可以是正值也可以是负的非零值。该算法需要利用冗余数方案（Avizienis 1961），从而可以允许数字的值取 0，−1

或 1。SRT 最重要的特点是每次迭代中不需要完全精确的比较，因此性能更好。

假设要计算一个数 R 的二次方根，S_i 表示第 i 次迭代后所得到的中间结果。第 i 步缩放的余数记为

$$Z_i = 2^i(R - S_i^2) \tag{3-8}$$

式中，$1/4 \geqslant R > 1$，$1/2 \geqslant S < 1$。从前面的余数计算可以推导出迭代表达式（McQuillan 等 1993）。

$$Z_i = 2Z_{i-1} - s_i(2S_{i-1} + s_i 2^{-i}) \quad i = 2,3,4\cdots \tag{3-9}$$

式中，s_i 表示第 $i-1$ 次迭代中的根数字。

通常，初始值 Z_0 设为 R，而平方根初始估计值 S_1 设为 0.5（受 R 的初始范围所限）。

虽然设计了大基数的二次方根算法（Ciminiera 和 Montuschi 1990，Cortadella 和 Lang 1994，Lang 和 Montuschi 1992），但对于很多采用大于 2 的基数的算法来说，需要通过 LUT 来获得二次方根的初始估计。后面的章节会涉及。

2. 基于函数迭代的二次方根算法

与 3.4.3 节的逼近除法相似，二次方根计算也可以利用函数迭代来完成，增加法和相乘法都可以。如果是增加法，则每次迭代都基于加法，并且每次迭代中退出计算的位数都相同。换句话说，这类方法线性地逼近最终解。CORDIC 算法可归到此类，它常用于实现矩阵对角化过程中的 Givens 旋转（Hamill 等 2000）。相乘法将每次迭代结果的精度提高了一倍，即它们二次收敛到结果，因此是很有意义的可选方案。然而，缺点是由于每次迭代中的乘法而增加了计算复杂度。

和除法算法相似，二次方根也可以用牛顿－拉普森或序列逼近算法来估计。采用牛顿－拉普森方法的迭代算法可以记为

$$x_{i+1} = X_i - \frac{f(x_i)}{f'(x_i)} \tag{3-10}$$

为此，我们需要选择一种合适的 $f(x)$，其根在解附近。一种可能的函数是 $f(x) = x^2 - b$，由此可得到式（3-11）所表示的迭代式。

$$x_{i+1} = \frac{1}{2}\left(X_i - \frac{b}{x_i}\right) \tag{3-11}$$

式（3-11）的缺点是要做除法。另一种替代方法是求二次方根的倒数，即 $1/x^2$。此时，利用 $f(x) = 1/x^2 - b$ 可得到式（3-12）所表示的迭代算法。

$$x_{i+1} = \frac{x_i}{2}(3 - bx_i^2) \tag{3-12}$$

通过式（3-12）求解之后，乘以原始值 X 即可得到二次方根，即 $1/\sqrt{X} \times X = \sqrt{X}$。

另一种计算二次方根函数的方法是使用序列逼近（Soderquist 和 Leeser 1995），即 Goldschmidt 算法，其表达式和除法的表达式相近（Even 等　2003）。

算法的目的是计算连续迭代后的结果，使得一个值逼近 1，而另一个值逼近最终结果。假设值 a，为了计算其二次方根，通过式（3-13）和式（3-14）迭代。

$$x_{i+1} = x_i \times r_i^2 \tag{3-13}$$

$$y_{i+1} = y_i \times r_i \tag{3-14}$$

令初始值 $x_0 = y_0 = a$，r 值由式（3-15）计算。

$$r_i = \frac{3 - y_i}{2} \tag{3-15}$$

$x_i \to 1$ 且因此 $y_i \to \sqrt{a}$。换句话说，经过不断迭代，x 逐渐逼近 1，而 y 逐渐逼近 $\sqrt{a}$。

与其他逼近的示例一样，算法充分利用了初始估计值 $1/\sqrt{a}$ 对初始值 x_0 和 y_0 预分频。

上面讨论的除法和二次方根逼近算法的示例中，都可以通过 LUT 得到期望解的初始值，从而大幅提高性能。我们将在下一节讨论。

3. 初始逼近技术

利用 LUT 得到的初始逼近值可以显著减少逼近算法的迭代次数。比如，产生除数 D 的倒数的逼近值 R_0 最简单的方式是直接从 LUT 中读取 $1/D$ 的逼近值。n 位的输入值 D 的前 m 位用来对 p 位的表进行寻址，该表存有倒数的逼近值。确定表中所存的值需要考虑将 D 从 n 位截短为 m 位时所产生的最大和最小误差。

LUT 的读取时间比较小，因此它可以快速得到解的部分位。然而，随着用来对查找表寻址输入值的增大，查找表的规模呈指数增长。对于 m 位寻址且输出为 p 位的表，它将有 $2m$ 条 p 位的表项。因此，LUT 的规模会迅速增大，从而增加读取时间。

通过合理组合 p 和 m，可以用尽可能小的表来获得所需的逼近精度。定义 p 与 m 的差值作为保护位数 g，则总误差可以表示为（Sarma 和 Matula　1993）

$$E_{\text{total}} = 2^{m+1}\left(\frac{1}{2^{g+1}}\right) \tag{3-16}$$

表 3-3 举例说明了不同 g 和 m 值时的逼近精度。这些结果有助于判断增加少量保护位能否额外提供足够的精度，同时需要权衡从 $m \sim m+1$ 每增加 1 位使表的规模增加一倍所付出的代价。

表 3-3 逼近精度示例

地址位	保护位	输出位	最小精度
m	0	m	$m+0.415$ 位
m	1	$m+1$	$m+0.678$ 位
m	2	$m+2$	$m+0.830$ 位
m	3	$m+3$	$m+0.912$ 位

另一种简单的逼近技术是 ROM 插值方法。该方法不是简单地将寄存器中的值截短到 m 位，而是将第一个不可见的位，即第（$m+1$）位置为1，其余所有较低有效位全部置为0（Fowler 和 Smith 1989）。这样做可以平均误差。然后在超出表中输出位的第1位上加1，再将得到的值四舍五入到表中条目的最低有效位。这种技术的优点是简单，但是它不适合较大的初始逼近值，因为它不会减小表的规模。

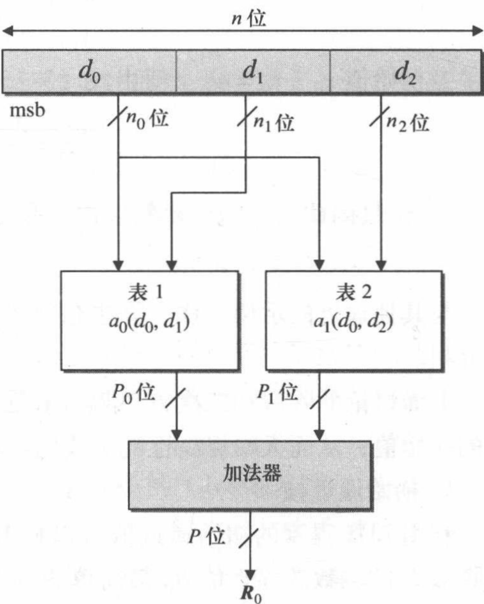


图 3-9 二分逼近法模块图（Schulte 等 1997）

人们开发了很多压缩表规模的技术，比如二分表，它利用两个或多个 LUT，然后将两个结果相加来得到逼近值。为了利用二分表来求倒数的逼近值，输入操作数被分成三部分，如图 3-9 所示。

前（n_0+n_1）位给出了在第一个 LUT 的位置，用系数 a_0 表示，有 p_0 个位。d_0 和 d_2 部分共有（n_0+n_2）位，给出了在第二个 LUT 中的位置，用系数 a_1 表示，有 p_1 个位。两个表的输出结果相加得到用二项泰勒级数展开式逼近的倒数 R_0。这样做的目的是利用前（n_0+n_1）个最高有效位给出第一个表中的位置，该表保存与 d_2 范围内的中值相加之后得到的系数。第二个系数的计算是将 d_0 和 d_2 部分与 d_1 范围内的中值相加得到。这种技术构造了一种平均方法，因此减小了截断误差。倒数逼近的系数通过式（3-17）计算。

$$a_0(d_0,d_1)=f(d_0+d_1+\delta_2)=\frac{1}{d_0+d_1+\delta_2} \tag{3-17}$$

$$a_0(d_0,d_1)=f'(d_0+d_1+\delta_2)(d_2=\delta_2)=\frac{\delta_2-d_2}{(d_0+\delta_1+\delta_2)^2} \tag{3-18}$$

式中，δ_1 和 δ_2 是常数，分别等于 d_1，d_2 的最大值和最小值之间的值。

上述方法的好处是用两个小的 LUT 比一个大 LUT 占用空间更小，却可获得相同的精度，即使把加法考虑进去也是一样。人们开发了简化二分逼近法的技术，其中一种是在两个表中都存储冗余二进制倒数值的正值和负值部分来避免加法（Sarma 和 Matula　1995）。这些值结合少量编码的方式将一些低位做四舍五入，从而达到最低有效位所要求的精度。这种编码方式只需很少的逻辑运算就可以将冗余二进制值转换为符合 Booth 编码的乘法器要求的操作数，也就是说，它们被转换为 Booth 编码的操作数。

3.5　定点和浮点的比较

如果我们坚持“更精确总是最好的”，那么毫无疑问应该采用浮点方法来表示数字。然而，浮点表示的芯片面积开销是 FPGA 实现中难以承受的，对我们前面提到的一些 DSP 应用，浮点表示所占的芯片面积开销超过定点表示的 10 倍（Lightbody 等　2007）。下面我们以图 3-10 所示的浮点加法器（Pillai 等　2001）为例进行分析，它是一种常用的加法器实现方法。首先，可以看到有好几个加法器/减法器及桶形移位器和其他控制逻辑。这个额外的逻辑被用来完成加法器实现中的多个归一化步骤。表 3-4 列出了所要求的多种功能，包括电路 A、电路 B 和电路 C 的旁路（图 3-10 中虚线所示部分）。表 3-4 中简单解释了两个操作数的指数值 e_1 和 e_2 与主位宽 p 在不同条件下所需的电路。需要注意的是，这里还没考虑特殊情况，比如 0 和操作数加减运算及操作数与无穷数加减的情况，因为我们只是想用这个示例来说明复杂度，以及分析为什么浮点硬件的开销远大于定点硬件的开销。

表 3-4　指数表示的浮点加法器运算（Pillai 等　2001）

指数标准	激活电路
$\lvert e_1-e_2\rvert>p$	C
$\lvert e_1-e_2\rvert\leqslant p$ 减	B
$\lvert e_1-e_2\rvert\geqslant p$ 加	A

针对浮点运算的芯片面积比较复杂，因为乘法器和加法器面积之间的关系发生了变化。在定点运算中，乘法器通常认为比加法器的面积大 N 倍，其中 N 表示字长。然而，在浮点运算中，浮点加法器所占面积和浮点乘法器所占面积相当，破坏了算法设计的初衷——多用加法而减少乘法的数量。表 3-5 从 Lightbody

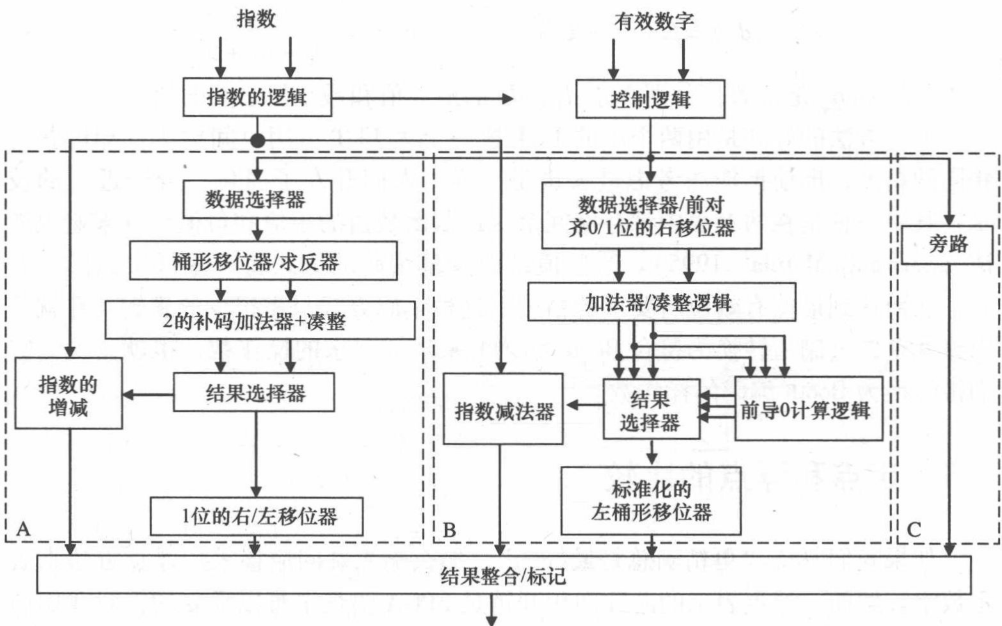


图 3-10　三路数据通道的浮点加法器模块图

等（2007）摘取了一些面积与速度的数据，与 Xilinx Virtex 4 FPGA 上实现的浮点加法和乘法有关。同时也列举了浮点除法，即与倒数函数有关的数据。这些值基于东北大学（North Eastern University）的可变精度浮点模块。

表 3-5　Xilinx Virtex 4 FPGA 上实现的多种浮点运算的面积和速度数据

功能	DSP48	LUT	触发器	速率/MHz
乘法器	4	799	347	141.4
加法器	×	620	343	208.2
倒数	4	745	266	116.5

决策过程（硬件选型）比较复杂，不仅仅是简单的面积和速度的比较，而应该基于实际应用要求来判断。比如，很多应用都有不同的数据字长，最终的精度也不同，见表 3-6。这些应用可能要求不同的输入字长，并且随着有限的内部字长变化，它们对误差的敏感度也不同。显然，较小的输入字长将有较小的内部精度要求，但对应用的洞察在决定内部字长需求时也起了很大的作用。人的眼睛对有限字长所表示的图像容忍度较高，尤其是当它们引起高频部分的失真时；然而，耳朵不能容忍任何频率的失真和噪声，尤其是高频部分。因此，在一些图像处理应用中可以采用简单的截短方法，在音频应用中却很少使用截短方法。

表 3-6 典型字长

应用	字长
控制系统	4 ~ 10
语音	8 ~ 13
声音	16 ~ 24
视频	8 ~ 10

表 3-7 列举了一些定点表示的动态范围估计。很明显，很多 DSP 应用可以用有限的字长，比如 12 ~ 16 位就能达到可接受的信噪比（Signal to Noise Ratio，SNR），具体字长依赖于所执行的内部运算稍有不同。由于 FPGA 中定点表示的性能增益大于浮点表示，因此定点实现已经逐渐成为主流，但两者之间的取舍还依赖于应用输入字长和输出字长、所要求的 SNR、内部计算复杂度及所执行的运算特点，即是否要进行类似矩阵求逆或迭代计算等特殊运算。

人们做了大量的工作来研究以最少的数字精度来满足性能要求。Canstantinides 等人（Constantinides 等 2004）从对设计质量的影响来考虑，寻求内部字长的精确位逼近。Fang 等人（Fang 等 2002）提出一种浮点设计流程，从算法生成浮点硬件，中间进行位宽优化，并且考虑硬件成本和功耗等问题。通常，设计者会利用一些工具中的定点库函数手动完成这个过程，前面提到的 MATLAB® 和 LabVIEW 都提供了相应的库函数。

表 3-7 固定字节动态范围

字节	字节范围	动态范围/dB
8	-127 ~ +127	$20\log 2^8 \approx 48$
16	-32768 ~ +32767	$20\log 2^{16} \approx 96$
24	-8388608 ~ +8388607	$20\log 2^{24} \approx 154$

3.6 总结

本章简要介绍了计算机算术运算的基础知识，讨论了实现基本算术运算函数所需的硬件，并介绍了一些复杂函数，如除法和二次方根运算。另外，本章概述了算术运算表示的关键因素，从硬件实现的角度讨论了定点和浮点选择需要考虑的问题。尽管目前 FPGA 支持浮点运算，但我们认为目前的 FPGA 技术非常适合定点实现，浮点运算还不太成熟。

参考文献

Andrews M (1986) A systolic sbnr adaptive signal processor. *IEEE Trans. on Circuits and Systems* 33(2), 230–238.

Avizienis A (1961) Signed-digit number representations for fast parallel arithmetic. *IRE Transactions on Electronic Computers* EC-10, 389–400.

Chandrakasan A and Brodersen R (1996) *Low Power Digital Design*. Kluwer, Dordrecht.

Ciminiera L and Montuschi P (1990) Higher radix square rooting. *IEEE Trans. Comput.* 39(10), 1220–1231.

Collange S, Detrey J and de Dinechin F (2006) Floating point or lns: choosing the right arithmetic on an aapplication basis. *Proc. 9th EUROMICRO Conf. on Digital System Design*, pp. 197–203.

Constantinides G, Cheung PYK and Luk W (2004) *Synthesis and Optimization of DSP Algorithms*. Kluwer, Dordrecht.

Cortadella J and Lang T (1994) High-radix division and square-root with speculation. *IEEE Trans. Comput.* 43(8), 919–931.

Ercegovac MD and Lang T (1990) Redundant and on-line cordic: Application to matrix triangularization. 39(6), 725–740.

Ercegovac MD and Lang T (1991) Module to perform multiplication, division, and square root in systolic arrays for matrix computations. *J. Parallel Distrib. Comput.* 11(3), 212–221.

Even G, Seidel PM and Ferguson W (2003) A parametric error analysis of Goldschmidt's division algorithm, pp. 165–171.

Fang F, Chen T and Rutebnbar RA (2002) Floating-point bit-width optimisation for low-power signal processing applications *Proc. Int. Conf. on Acoustics, Speech and Signal Proc.*, vol. 3, pp. 3208–3211.

Flynn M (1970) On division by functional iteration. *IEEE Trans. on Computing* C-19(8), 702–706.

Fowler D L and Smith JE (1989) High speed implementation of division by reciprocal approximation *IEEE Symp. on Computer Arithmetic*, pp. 60–67.

Hamill R, McCanny J and Walke R (2000) Online CORDIC algorithm and vlsi architecture for implementing qr-array processors. *IEEE Trans. on Signal Processing* 48(2), 592–598.

Heron JP and Woods RF (1999) Accelerating run-time reconfiguration on FCCMS. *Proc. IEEE Symp. on Field-Programmable Custom Computing Machines*, pp. 260–261.

Ito M, Takagi N and Yajima S (1995) Efficient initial approximation and fast converging methods for division and square root. *Proc. 12th IEEE Symp. on Comp. Arith.*, pp. 2–9.

Knowles S, Woods R, McWhirter J and McCanny J (1989) Bit-level systolic architectures for high performance IIR filtering. *Journal of VLSI Signal Processing* 1(1), 9–24.

Lang T and Montuschi P (1992) Higher radix square root with prescaling. *IEEE Trans. Comput.* 41(8), 996–1009.

Lightbody G, Woods R and Francey J (2007) Soft IP core implementation of recursive least squares filter using only multiplicative and additive operators *Proc. Int. Conf. on Field Programmable Logic*, pp. 597–600.

Ma J, Deprettere EF and Parhi K (1997) Pipelined cordic based QRD-RLS adaptive filtering using matrix lookahead *Proc. IEEE Intl. Workshop-SiPS*, pp. 131–140.

McQuillan SE, McCanny J and Hamill R (1993) New algorithms and VLSI architectures for SRT division and square root. *Proc. IEEE Symp. Computer Arithmetic*, pp. 80–86.

Muller JM (2005) *Elementary Functions, Algorithms and Implementation*. Birkhauser, Boston.

Obermann SF and Flynn MJ (1997) Division algorithms and implementations. *IEEE Trans. Comput.* C-46(8), 833–854.

Oklobdzija V and Ercegovac M (1982) On division by functional iteration. *IEEE Trans. Comput.* C-31(1), 70–75.

Omondi AR 1994 *Computer Arithmetic Systems*. Prentice Hall, New York.

Pillai RVK, Al-Khalili D, Al-Khalili AJ and Shah SYA (2001) A low power approach to floating point adder design for DSP applications. *Journal of VLSI Signal Proc.* 27, 195–213.

Robertson J (1958) A new class of division methods. *IRE Trans. on Electronic Computing* EC-7, 218–222.

Sarma DD and Matula DW (1993) Measuring the accuracy of ROM reciprocal tables *Proc. IEEE Symp. on*

Computer Arithmetic, pp. 95–102.

Sarma DD and Matula DW (1995) Faithful bipartite rom reciprocal tables. *Proc. IEEE 12th Symposium on Computer Arithmetic*, pp. 17–28.

Schulte MJ, Stine JE and Wires KE (1997) High-speed reciprocal approximations. *Proc. 31st Asilomar Conference on Signals, Systems and Computers*, pp. 1183–1187.

Sklansky J (1960) Conditional sum addition logic. *IRE Trans. Elect. Comp.* EC-9(6), 226–231.

Soderquist P and Leeser M (1995) An area/performance comparison of subtractive and multiplicative divide/square root implementations *Proc. IEEE Symp. on Computer Arithmetic.*

Soderstrand MA, Jenkins WK, Jullien GA and Taylor FA (1986) *Residue Number Systems Arithmetic: Modern Applications in Digital Signal Processing*. IEEE Press.

Takagi N, Asada T and Yajima S (1991) Redundant CORDIC methods with a constant scale factor for sine and cosine computation. 40, 989–995.

Tocher K (1958) Techniques of multiplication and division for automatic binary computers. *Quart. J. Mech. Appl. Math.* XI(3), 364–384.

Volder JE (1959) The CORDIC trigonometric computing technique. EC-8, 330–334.

Walther JS (1971) A unified algorithm for elementary functions. *Spring Joint Comput. Conf.*, pp. 379–385.

第4章 FPGA 技术概述

4.1 引言

DSP 实现的技术和硅工艺的显著发展密切相关。正如本书序言中所强调的，晶体管的成本在持续下降，有效利用晶体管已经成为新兴市场的主要驱动力，并且一直影响大量 DSP 技术的发展。得益于硅工艺的发展，不仅平台越来越便宜，并且速度更快、功耗更低，刺激了一批 DSP 应用市场的发展，尤其是移动通信技术和数字视频产品。

我们在第 2 章已经提到，数字域的系统有很多优点，尤其是精度可保证、可完美复制并且不容易老化，这些技术的发展是实现未来新系统的关键。在 20 世纪 60 年代末到 70 年代初，Leland B- Jackson 与其贝尔实验室的同事一起开发了最早的 DSP 滤波器电路（Jackson 1970）。那时候的主要目的是设计能完成基本滤波函数，如 FIR 和 IIR 等滤波器的芯片。人们观察到一个重要的特性，即晶体管的二进制操作非常适合实现 DSP 系统所需的基本数字运算。

从 20 世纪 60 年代开始，涌现了大量的技术，从简单的微控制器到专用 DSPSoC，前者的性能要求通常是采样速率，一般在中等的 kHz 量级上，而后者的性能接近每秒一万亿次操作（TeraOPS）的量级。对于处理器这一类型，人们开发了多种形式的架构，从单核处理器到多核处理器，专用 DSP 微处理器——包括专用硬件来高效地实现特殊 DSP 函数，还包括可重配 DSP 处理器架构。另一方向是专为特殊应用领域开发的专用指令集处理器（Application Specific Instruction Prcessor，ASIP）的发展。作者认为，DSP 系统实现的主要准则与所采用的电路架构有关。一般来说，硬件资源及它们之间如何连接对最终的 DSP 系统性能有重要影响。FPGA 可以根据算法的需求来设计架构，但会增加设计成本。本章将概括多种实现 DSP 系统的技术，将各种方案进行比较，并介绍一些相关示例。由于第 5 章将专门讨论各种 FPGA 架构，因此本章仅简单介绍 FPGA 的相关方案。本章讨论的主题包括可编程能力、编程环境（包括工具、编译器和体系结构）、在指定平台上针对特殊 DSP 函数的优化空间，以及与面积、速度、吞吐量、功率和鲁棒性有关的设计质量。本章结构如下：4.2 节将概述电路结构的一些问题，探讨了技术的性能局限及可编程能力的重要性。4.3 节将讨论 DSP 系统的功能需求，着重阐述计算复杂度、并行处理、数据独立性和算术运算优势等

问题。4.4 节将概述处理器的分类，4.5 节将简单介绍微处理器，4.6 节将介绍 DSP 处理器。4.7 节将介绍一些并行处理机，包括脉动阵列体系结构、单指令多数据流（Single Instruction Multiple Data，SIMD）和多指令多数据流（Multiple Instruction Multiple Data，MIMD），并列举了一些示例。为了完整起见，4.8 节简单回顾 ASIC 和 FPGA 的发展路线，本书后续章节将继续探讨。最后一节探讨如何对多种技术进行比较，为第 5 章将要讨论的 FPGA 进行铺垫。

4.2　架构和可编程能力

在很多处理器系统中，设计不仅仅表示完成必要的顶层代码，同时会考虑一些底层技术架构，目的是优化代码质量并提高性能。直白地说，就是牺牲性能来换取一定的可编程能力。比如，基于冯・诺依曼顺序处理模式的微处理器架构。其基本架构固定，通过将算法有效地调度到内部串行处理架构上来获得最佳性能。如果算法的计算过程本质上并行度很高（DSP 中常常出现的情况），则性能要打折扣。如果我们考虑另一种极端情况，开发一种基于 SoC 的架构来满足算法计算复杂度的需求，并且达到所需的并行度（如果可行的话），则最佳性能需要考虑面积、速度和功耗。为此，需要投入大量设计力量来确保硬件实现评估准则与算法性能标准匹配，从应用的角度来说就是最终的设计能正确工作。

为了更好地理解构建电路架构的思想，我们先回顾一下 1969 年的技术发展情况。当时，以晶体管数量来衡量的硬件处理能力，因此非常重要的处理过程，比如 Jackson（1970）介绍的滤波器不得不用串行方式实现。目前的 FPGA 技术能提供好几百位并行的乘法器，因此也就意味着不同类型的架构，算术运算类型及最终的性能都完全不同。其目的是充分利用可用的硬件资源而不仅仅是达到应用的性能指标。当 FPGA 技术因利用硬件资源来满足性能需求而大受欢迎时，其架构开发也面临一系列与架构构建过程有关的问题，比如设计时间、需要验证和测试架构的各种可能模式，以及与“一次就做对”的设计相关的问题。在特定硬件平台上实现算法可以在吞吐速率、延迟、芯片面积、能量、功耗等方面进行比较，另外还有一个可用来对比这些技术的重要主题是可编程能力，严格地说是易于实现的可编程能力。本章后续小节将进一步介绍，DSP 硬件架构的可编程能力可在一定程度上调整。针对平台的固有特点，人们已经开发了高效的软件编译器来生成最有效的实现，因此可以用高级软件语言编程来实现简单的固定硬件架构平台。然而，由于平台越来越复杂且灵活，影响了这些工具的复杂度和效率，因此，需要引入特殊的指令来实现可编程性。编译器的主要目的就是理解那些不是专门针对特定硬件架构所生成的源代码，并且知道如何利用这些特殊的函数来改进性能。粗略地看，我们认为构建可编程的电路架构在性能上具有优势，但在

架构演进上还面临一些问题，不管是为了满足应用需求的小变化还是适配相似的应用。这也突显了设计工具和环境的重要性，我们将在第 11 章进行描述。

从可编程能力角度看，ASIC 是另一种方案，硬件平台可能被大规模开发来满足系统或某些特殊领域标准应用的需求。比如，基于 WCDMA 的移动电话要求专用的标准 DSP 功能，可通过开发包含处理器和专用硬件 IP 模块的 SoC 平台来实现。对大多数移动电话实现方案来说，这是满足能量要求的关键因素。然而，硅工艺的成本现在已经将 ASIC 实现推向特殊领域，通常，这一领域的解决方案要么是大量生产，要么具有特殊的要求，比如在一些传感器中要求超低功耗。

开发同时具有一定程度的硬件可编程能力和软件可编程能力架构的思想已经在 FPGA 架构上实现了。FPGA 架构主要是由逻辑元素、LUT、寄存器、布线、可配置 I/O 和一些专用硬件组成，它是实现高性能的理想架构。

4.3 DSP 功能特点

通常，DSP 运算有如下特点：计算量大，适合采用并行处理器实现，并且并行化程度高；数据独立，在一些情况下相对科学计算等高性能应用来说对算术运算要求较低。为了能够判断这些特点对 DSP 算法在 FPGA 等硬件平台上实现时的影响，我们需要充分理解这些特点。

1. 计算复杂度

DSP 算法可能很复杂。比如，考虑式（4-1）的 N 抽头 FIR 滤波器表达式（第 2 章中的式（2-11））。

$$y_n = \sum_{i=0}^{N-1} a_i x_{n-i} \tag{4-1}$$

从上面的计算式可看到，a_0 必须和 x_n 相乘，接着是 a_1 和 x_{n-1} 相乘，再相加，以此类推。假设抽头数为 N，这意味着计算 y_n 需要做 N 次乘法和 $N-1$ 次加法。

$$y_n = a_0 x_n + a_1 x_{n-1} + a_2 x_{n-2} + \cdots + a_{N-1} x_{n-N+1} \tag{4-2}$$

假设另一次计算从下一个样点 x_{n+1} 输入开始，我们可以定义每周期要求的计算量，即每个样点 $2N$ 次运算或每个抽头两次运算。如果用处理器来实现，要求每个周期装载一次数据，则可能需要 2 ~ 3 个周期来装载数据和系数并存储累加和。这可能意味着每个周期额外需要 3 次运算，因此最终每个抽头上需要 6 次运算，从总体来看，每个样点需 $6N$ 次运算。对于采样速率为 44. 2kHz 的音频应用，128 抽头的滤波器要求 33. 9 百万样本/秒（MSPS）的处理速度，对一些技术来说可能是能实现的，但如果考虑 13. 5MHz 的图像处理速度，则计算速度会暴增，可能达到每秒 10 千兆样本（GSPS）的处理速度。此外，这可能只是系统

中一个函数的需求，计算量也只是全部计算量的一小部分。

对于处理器的实现，设计者会判断硬件能否满足吞吐量的需求，一般是通过将处理器的时钟速度除以每个周期需要完成的运算总数来决定的。这种方法有可能不能很好地发挥系统性能，例如，如果 N 较大，则时钟速度和吞吐率的差异也会较大。时钟速率可以很快，以便提供必要的采样速率，但在系统设计时面临的问题是：很快的时钟速率与控制功耗，尤其是动态功耗相矛盾，因为功耗直接和时钟速率有关。

2. 并行化

DSP算法的特征是并行化程度很高。比如，式（4-1）的表达式可以在单个处理器实现，或者采用并行方式实现，如图4-1所示，图中的每个元素都是一个硬件单元，意味着有127个寄存器对应延迟元素，127个乘法器用来计算 $a_i x_{n-i}$ 的乘积（其中 $i=0, 1, 2, \cdots, N-1$），还有128个输入的加法，它通常可用加法器树来实现。这种方式是通过增加硬件复杂度来将算法的迭代过程在一个样本周期中实现。显然，高度并行化的系统和存储能力可以加快计算。还有其他一些并行化方法用来达到所要求的性能，本书后续章节中将重点介绍。

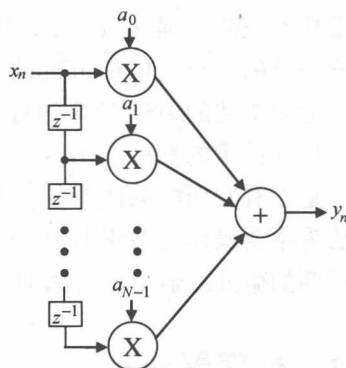


图4-1 简单的FIR滤波器并行实现结构

3. 数据独立性

数据独立性的特点非常重要，它为计算的排序提供了条件。对减少寄存器和数据存储器的需求来说非常重要。比如，考虑式（4-1）中FIR滤波器的 N 次迭代计算。

$$
\begin{aligned}
y_n &= a_0 \underline{x_n} + a_1 x_{n-1} + + a_2 x_{n-2} + \cdots + + a_{N-1} x_{n-N+1} \\
y_{n+1} &= a_0 x_{n+1} + a_1 \underline{x_n} + + a_2 x_{n-1} + \cdots + + a_{N-1} x_{n-N+2} \\
y_{n+2} &= a_0 x_{n+2} + a_1 x_{n+1} + + a_2 \underline{x_n} + \cdots + + a_{N-1} x_{n-N+3} \\
&\vdots = \vdots \\
y_{n+N-1} &= a_0 x_{n+N-1} + a_1 x_{n+N} + + a_2 x_{n+N+1} + \cdots + + a_{N-1} \underline{x_n}
\end{aligned}
$$

我们可以看到，上述 N 次计算都需要数据 x_n，因此我们完全可以用数据 x_n 并行地执行从 y_n 到 y_{n+N-1} 这 N 次计算，从而避免存储 x_n。显然，现在需要存储中间累加器的结果。这给设计者提供了多种途径来优化系统性能，并且最终的设计也有不同选项。上面仅仅是数据独立性的一个方面。

4. 算术运算要求

在很多DSP技术中，对输入数据字长的要求是尽可能降低内部精度。以表

3-6 中不同应用的可变字长为例，通常，输入字长由模数转换器（A-D）的精度或原始信号中的噪声决定，后者可能会影响系统中的总噪声。内部字长依赖于计算的类型，比如乘法或加法及其数量，其增长可以限制，这就是说合适的定点实现就已经足够了。

较低的算术运算需求至关重要，尤其是对 FPGA 实现，因为浮点缺乏灵活度。有限的字长意味着较小的寄存器需求、更快的实现——因为加法器和乘法器的速度由输入字长决定，以及较小的面积。因此，设计人员花了很多精力研究如何选定最大字长。其中有趣的一点是，对很多处理器实现来说，外部和内部字长在架构开发时就提前确定了，但对 FPGA 来说，需要详细分析来决定在 DSP 系统不同部分的字长（Constantinides 等 2004）。

上面描述的 DSP 计算的特点对确定一种高效的实现方案非常重要，有的时候还推动技术的发展。比如，早期 DSP 处理器和微处理器之间的一个主要区别在于是否有专用的乘法器核。对 DSP 处理器来说是切实可行的，因为就是为以乘法为主要操作的 DSP 应用设计的，但对一般的处理应用来说就不一样了，因此早期的微处理器中没有添加乘法器。

4.4 处理器分类

实现 DSP 的技术包括微控制器和单芯片 DSP 多处理器，而后者又包括将超长指令字（Very Long Instruction Word，VLIW）扩展以支持指令并行化的传统处理器架构和专门为特殊应用领域设计的专用架构。虽然在 Flynn 提出的分类方法之后还有其他更详细的分类法，但 Flynn 的分类方法是使用最多的，他将指令和数据当作计算机中两个独立的流，参见表 4-1 的分类总结。

表 4-1 Flynn 提出的处理器分类方法

类型	描述	示例
单指令单数据（SISD）	单指令流操作单数据流	冯·诺依曼处理器
单指令多数据（SIMD）	多个处理单元，同步操作单数据流	超长指令字处理器
多指令单数据（MISD）		没有实际示例
多指令多数据（MIMD）	多个处理单元独立操作各自的数据流	多处理器

4.5 微处理器

图 4-2 所示为经典的冯·诺依曼（von Neumann，vN）微处理器架构。这种类型的架构是典型的单指令单数据（Single Instruction Single Data，SISD）类型架

构，串行地评估一串指令并按顺序对特定数据执行指令。这种架构由5个单元组成：存储数据和指令的存储器、指令获取和译码（Instruction Fetch and Decode，IFD）单元、算术逻辑单元（Arithmetic Logic Unit，ALU）和内存访问（Memory Access，MA）单元。这些单元对应处理过程的4个不同阶段，机器执行每条指令都会重复这些过程：

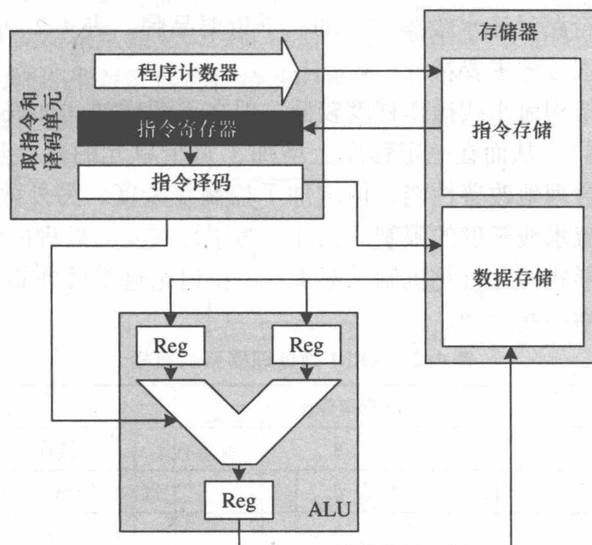


图4-2 冯·诺依曼微处理器架构

1）指令获取；
2）指令译码；
3）执行；
4）内存访问。

在指令获取（Instruction Fetch，IF）阶段，IFD单元将程序计数器（Program Counter，PC）所指地址中的指令装载到指令寄存器（Instruction Register，IR）。在指令译码（Instruction Decode，ID）阶段，IFD将指令译码并为ALU生成操作码及两个操作数的地址，它们将被送到ALU的输入寄存器。在执行阶段，ALU对操作数执行操作码所指定的运算，得到结果后在MA阶段将其返回寄存器。通常，这类SISD机器可以根据指令集类型再分成两类：①复杂指令集计算机（Complex Instruction Set Computer，CISC）采用复杂指令格式，可以为特殊运算高度定制，其代码比较紧凑，但会导致流水线指令执行变得复杂；②精简指令集计算机（Reduced Instruction Set Computer，RISC）采用标准的简单指令格式，可通过常规方式处理，流水线操作可提高吞吐量，但会增加代码量。vN处理器架构是为通用计算而设计的，由于处理流程高度串行的特点，因此只局限于嵌入式

应用，这种处理器架构适合于通用目的环境。然而，嵌入式处理器必须要灵活，它们通常要针对特殊应用进行调整，且性能要求较高，比如低功耗或高吞吐量。

4.5.1 ARM 微处理器架构系列

ARM 系列嵌入式微处理器是 RISC 处理器架构的一个典范，它是 RISC 处理器架构（指令执行路径流水线操作）的一个重要品牌。表 4-2 列举了 ARM 微处理器系列中 3 个型号的主要特征。

这些处理器架构流水线操作程度较低，但在逐渐增加（如表 4-2 和图 4-3 列出的 ARM 处理器），从而在一定程度上增加了每个单元的吞吐量。随着流水线长度增加，可以合理地改善性能，但增加了控制复杂度，另外限制因素也增加，其中一个就是对流水线长度的限制。因此，为了增加实时处理的性能，处理器架构必须考虑其他形式的并行化机制。第 4.6 节将讨论这类处理器架构中采用的不同的技术，并列举一些示例。

表 4-2 ARM 微处理器系列概括

部件	管道深度	简介
ARM7	3	读取、译码、运行
ARM9	5	读取、译码、ALU、访问内存、回写
ARM11	8	见图 4-3

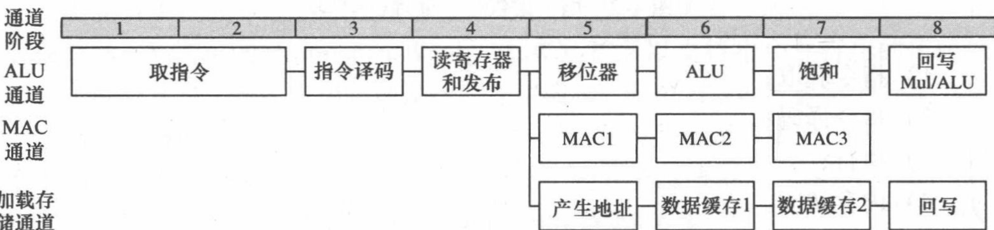


图 4-3 ARM11 流水线结构

4.6 DSP 微处理器

第 4.5 节提到微处理器架构串行处理的特点，使其不适合有效地实现需要复杂计算的 DSP 系统，要么就是不能达到所要求的采样速率，要么就是能够满足采样速率需求但却消耗大量功率。对微处理器的实现来说，串行架构主要用于数据处理应用，在计算过程中，很多晶体管都起不到实际作用。因此，固定的通用处理器付出的代价是，很多晶体管消耗能量但却对性能毫无贡献。

因此，人们开始寻求其他适合 DSP 的处理器架构。在 20 世纪 80 年代，TI

公司设计的 DSP 微处理器 TMS32010 上市，它的功能和微处理器相似，但不同的是，它基于哈佛架构，将程序存储器和数据存储器分开，并且使用独立总线。概括地说，它们是针对 DSP 需要执行乘法和累加运算而优化的微处理器架构，功耗更低。图 4-4 所示为冯·诺依曼架构和哈佛架构的区别。在冯·诺依曼架构中，程序和数据使用同一块存储器，因此性能瓶颈在存储器。在哈佛架构中，数据存储器和程序存储器相互独立，因此数据和程序都独立地加载到处理器。哈佛架构还有一些专用硬件来对数据进行特殊的操作。起初是专用的乘法器，随着集成度的增加，后来增加了更多复杂的功能。

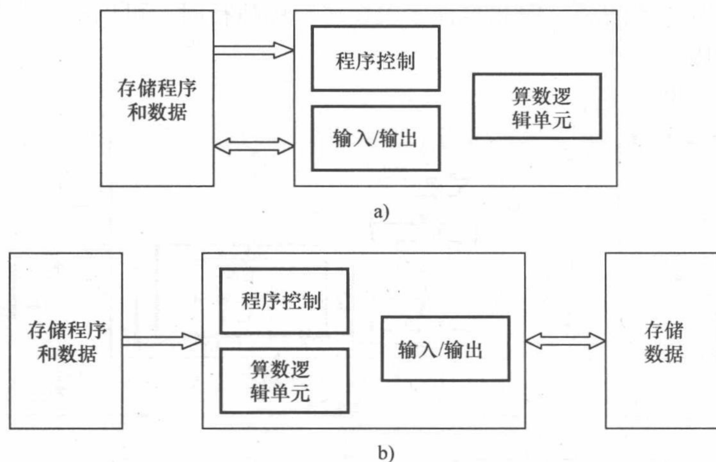


图 4-4　冯·诺依曼架构和哈佛架构

a) 冯·诺依曼架构　b) 哈佛架构

独立的数据存储器和程序存储器，以及专用 DSP 硬件是早期 DSP 处理器的重要基础。TI 公司设计的 TMS32010 DSP 被认为是第一个 DSP 架构，也是哈佛架构早期的应用之一，充分体现了哈佛架构的特点。从图 4-5 可以清楚地看到，其结构中包括程序和数据总线，虽然是早期设备，但除了常规的 ALU，它还包含一个专用的乘法 - 累加函数。较早的 TMS32010 16 位处理器指令周期为 200ns（5MIPS），可以在 400ns 内完成一次乘 - 累加（Multiply-Accumulate，MAC）操作。

此后，基于这个架构做了很多改进（Berkeley Design Technology 2000），下面将简单介绍。TI 公司的 TMS320C64xx 系列器件将在下一节介绍（Dahnoun 2000，Texas Instruments Inc. 1998）。

（1）VLIW。从早期的处理器架构到现代处理器架构，内部总线字长在逐渐增加。它利用多个处理功能单元，使得每条指令的多个运算可以并行执行。这种处理器还可以继续挖掘这些硬件单元的潜能，当然，这取决于要执行的计算及基

于此架构的编译器的效率。当技术开始向高级编程语言发展时，问题变复杂了，高级编程语言需要对编译器做很好的优化，从而可以有效地翻译高级语言的代码并清除程序员造成的冗余编码。

（2）数量增加的数据总线。在一些器件，比如亚德诺半导体（Analog Device，ADI）的超级哈佛架构（Super Havard Architecture，SHARC）ADSP-2106x（ADI，2000）中，增加了数据总线的数量。其依据是很多 DSP 运算都有两个操作数，因此需要将 3 块信息（包括指令）输入到存储器。通过增加总线数量可以提高运算速度，但同时也增加了器件的引脚数。然而，SHARC 架构通过使用程序缓存来避免此问题，因此当程序未在缓存中执行时，可以将指令的总线当作数据总线使用。

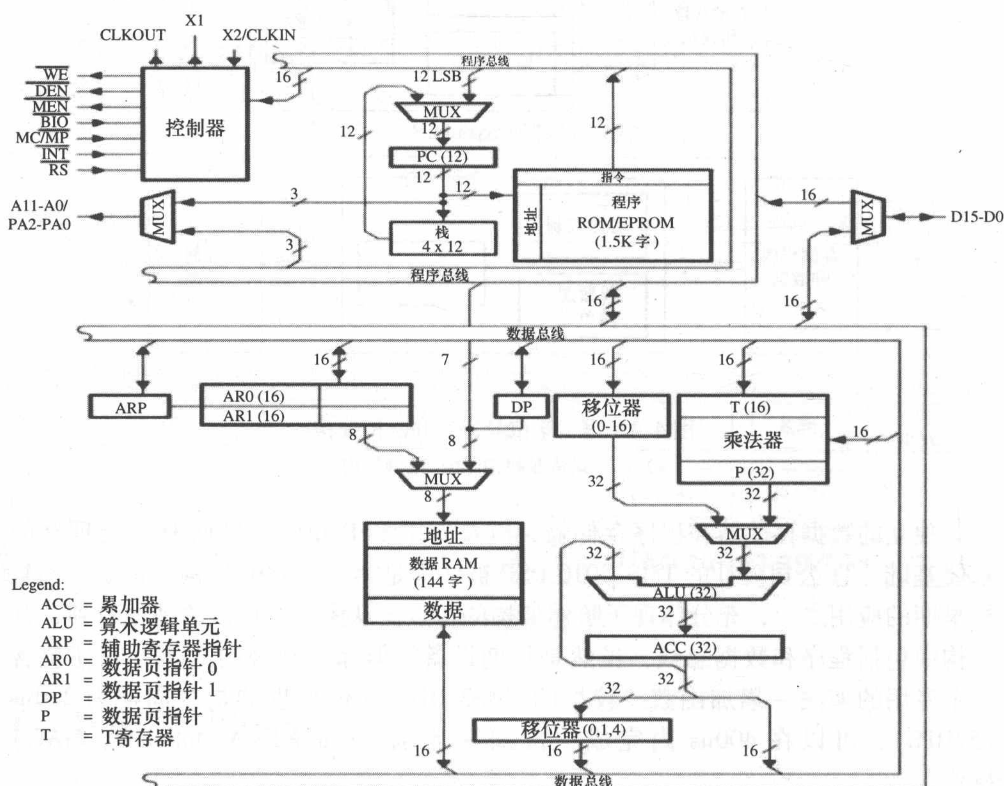


图 4-5 TI 公司的 TMS32010 DSP 处理器（由 TI 公司许可复制）

（3）流水线引入。VLIW 的目的是并行化，另一种开发并行化的方法是引入流水线，两者都在 DSP 架构的处理单元内，针对程序执行过程优化。流水线作业将处理过程分成小的时间单元，从而可以在相同的硬件上同时执行几个时间点的重叠计算。其代价是增加了延迟。流水线也可以应用在处理器控制单元内，控

制程序获取、指令分发和指令译码运算，如图 4-6 所示，具体细节将在下一节介绍。

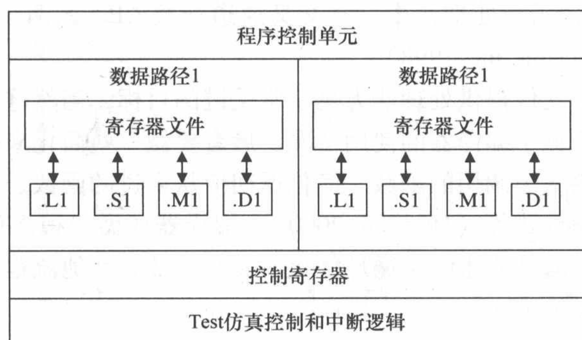


图 4-6 TI 公司的 TMS320C62 和 TMS320C67 模块图（Dahnoun 2000）

（4）定点运算。实际应用中，很多 DSP 系统只需要完成定点算术运算，并不要求 DSP 处理单元提供全精度的算术运算。因此，人们为不同的应用环境开发定点或浮点的应用。然而，即使是定点应用，部分运算也不需要一些处理器（如 TMS320C64xx 系列处理器的 32 位精度）所提供的全部定点精度范围，从而使得利用率较低。举例来说，对图像处理应用中的滤波器，输入字长可能在 8 ~ 10 位，系数字长可能在 12 ~ 16 位。因此，乘法阶段并不要求任何大于 16 × 16 的乘法器，DSP 处理器就可以利用这一特点来管理其中的处理单元，比如 TMS320C64xx 中的处理单元就能够在同一时间完成两个 16 × 16 位的乘法，从而提高吞吐量。此时，处理器并没有因内部所使用的字长而降低效率。

上述的优化方案一直发展了好多年，也提高了性能。然而，为了理解架构在某些应用中的实现，还需要考虑运算本身。

4.6.1 DSP 微运算

本节将介绍 TMS320C64xx 系列架构（Dahnoun 2000，Texas Instruments Inc. 1998），这是一款经典的 DSP 处理器。图 4-6 所示为该处理器 CPU 部分的简化模型。图中的程序控制单元（Program Control Unit，PCU）先获取指令，将指令分发到对应的处理器，然后执行指令译码。TMS320C64xx 系列器件中，有两个数据通道单元。在 TI 公司的器件中，将功能单元分成两组，每组有 4 个功能单元（L，M，S 和 D），其中 L 单元可用于 32/40 位的算术运算和比较运算、32 位的逻辑运算、归一化和位统计运算，以及 32/40 位的饱和算术运算，M 单元可以执行多种类型的乘法运算，比如 4 个 8 × 8、2 个 16 × 16 和 1 个 16 × 32 的运算，S 单元可以执行多种算术运算，即移位、分支和比较运算，D 单元完成各种

加载和存储操作。需要说明的是，这不是处理单元的精确定义（Texas Instruments Inc. 1998），但给出了功能的概念。该处理器还包括寄存器堆和模块之间通信的通道。在 TI 公司的处理器中，有交叉通道连接 CPU 的两个部分（Dahnoun 2000，Texas Instruments Inc. 1998）。

能够利用多个硬件提供处理能力是非常关键的目标，当然这不仅依赖于将要执行的计算，还依赖于编译器的使用情况，后者要做一些简化来提高效率。这些简化流程包括移除从未调用的函数、简化返回值从未过的函数、重新调整函数声明和函数变量传输方式等（Dahnoun 2000）。编译器还做了很多优化来利用基础架构的优点，包括软件流水线、循环优化、展开循环和其他流程来移除全局分配和表示（Dahnoun 2000）。

4.7 并行机

串行模型目前能够实现很多算法，从这种意义上来说已经相当不错了，但 DSP 实现真正的增益来自硬件并行化。因此，人们投入大量精力开发硬件，其中包括由早期的晶体计算机演进而来的并行硬件（Inmos 1989）。然而，获得这种并行化程度正是关键问题。脉动阵列（Kung 和 Leiserson 1979，Kung 1988）是实现这种并行化的一种重要架构，本节首先将介绍该架构。

4.7.1 脉动阵列

1979 年，Kung 和 Leiserson 将脉动阵列架构引入超大规模集成电路（Very Large Scale Integration，VLSI）（Kung 和 Leiserson 1979）。总体上说，它们具有下面的特征（Kung 1988）：

1）一组高度并行化的处理器；

2）处理器类型数量较少；

3）控制简单；

4）局部的互连接。

其处理能力来自于并行使用很多简单的单元，而不是几个非常强大的单元串行使用。这种架构非常适合具有简单且常规数据流的并行算法，比如矩阵运算。通过流水线，脉动阵列中的运算可以连续地在阵列中推进，从而能够充分利用处理单元。

图 4-7 所示为一个简单的线性结构脉动阵列示例。其中，黑色圆点表示每个处理单元（Processing Element，PE）后的流水线阶段。穿过这些流水线阶段的线条表示调度线，它表示同时处在相同迭代过程的 PE。换句话说，这些计算在同一时钟周期执行。

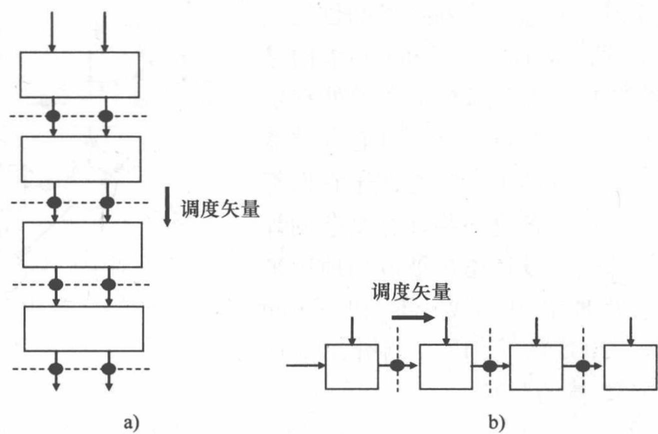


图 4-7　线性结构脉动阵列架构
a）列　b）行

图 4-8 所示为另外两个示例。图 4-8a 所示为多个单元组成的方形阵列，每个单元都使用局部互连接。这种类型的阵列非常适合矩阵运算。图 4-8b 所示为一个六边形结构的脉动阵列示例。上述示例中，每个 PE 都只从最近的单元接收数据，每个处理器都包含一个小型的局部寄存器用来存储中间结果。数据在阵列中的传递通过同步时钟来控制，像水泵一样推动数据在阵列中移动，由于和心脏将血液泵送到全身的过程相似，“脉动”因此而得名。图 4-9 所示为用于 QR 分解的脉动阵列。该阵列由两类单元组成，即边界单元和内部单元，都是局部互连接。

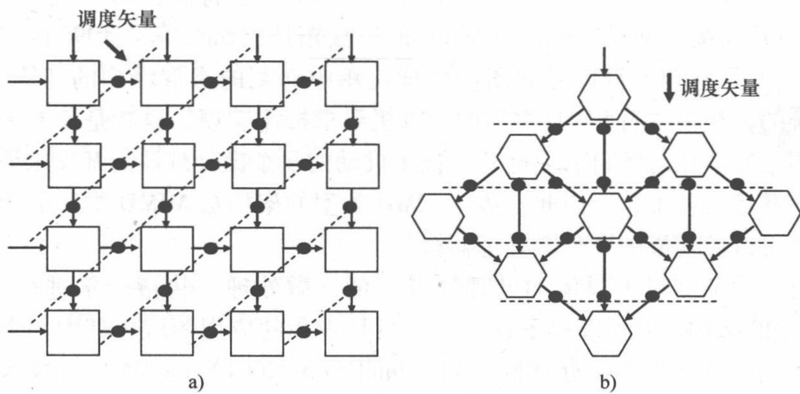


图 4-8　脉动阵列架构
a）方形　b）六边形

脉动阵列的思想有很多种应用方式。McCanny 和 McWhirter（1987）将其应

用到位级脉动阵列，而脉动阵列技术的创始人将此思想进一步开发成iWarp，Intel和卡内基梅隆大学在1988年尝试用其在单个微处理器上搭建完整的并行计算节点，并且具有存储器和通信链路。此类开发的主要问题在于它们都是针对特定应用，用来解决一些计算复杂的算法；相反，脉动阵列的设计思想被成功地用来开发大量信号处理芯片（Woods 和 Masud 1998，Woods 等 2008）。第12章将介绍如何将这一思想用于IP核的开发。

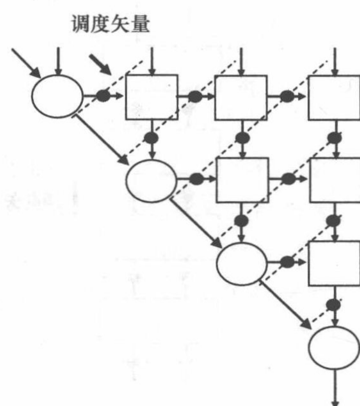


图4-9 三角QR脉动阵列

4.7.2 SIMD架构

第一类基于SIMD架构的并行计算机有Illiac IV（Barnes 等 1968）和第二代互连机器（Connection machine-2，CM-2）（Hillis 1985）。Illiac IV表示伊利诺斯积分仪和自动计算机，它是一种高度并行的机器，具有64个处理单元，全部由一个控制单元（Control Unit，CU）控制。这些处理单元同时执行相同的操作，每个处理单元都有各自的存储器。最初的CM-2思想来自于麻省理工学院，它是一个由简单处理单元组成的超立方体，处理单元又由具有各自存储器的简单CPU组成。CM-2有很多处理单元（通常为64000个），每次处理一位，但后来扩展到浮点运算。这些早期的架构都可以归为通用SIMD类型的架构，其中由一个CU获取指令并且将指令分发给多个处理单元，它们同时执行相同的操作（Sima等 1997）。区分SIMD类型中的各种架构的主要特征有PE复杂度、PE自主程度、PE连接类型和数量，以及PE的连接拓扑（Sima 等 1997）。基本上，SIMD模型非常适合大量信号和图像处理应用中遇到的常规结构的计算。然而，开发传统的大规模并行SIMD类型的计算机要求按需实现，也就是说专为某些算法或应用定制。相对于目前受硅芯片技术推动的复杂微处理器的开发来说，其成本很高，并且收益不大。因此，传统SIMD类型的架构被MIMD类型的架构所取代，后者可以由现成的廉价微处理器构建。

最近，为了应对多媒体的处理任务，现代微处理器将指令集进行了扩展，SIMD类型的架构又开始获得重视，虽然只是很小的应用领域。其中一个范例是Intel最初用于其奔腾3微处理器系列（Intel Corp. 2001）的SIMD流技术扩展第二代（Streaming SIMD Extensions2，SSE2）；另一个用例是苹果、IBM和摩托罗拉联合设计的Altivec（Intel Corp. 2001），它可将128位的处理分解为并行的16个8位或者8个16位的整数运算，或者4个浮点运算。这些新指令集意味着要改动底层处理器的架构，包括处理流程和寄存器堆的组织，同时也增加了指令

集。另外，开发人员还设计了单芯片专用SIMD架构，比如Imagine处理器（Khailany等2001）和Clearspeed的CSX600（Clearspeed Tech. plc 2006）。针对这些平台的高效高级编程目前仍然是一个热点研究问题，下面将简要介绍这两种处理器。

1. Imagine处理器

Imagine芯片采用并行架构，它有48个浮点算术逻辑单元和一个特殊的分级存储器，针对程序流进行了优化（Kapasi等2002）。Imagine项目不仅包括架构开发，还包括基于“流编程”模型的编程环境开发。开发编程环境和平台的关键是实现一个高效的平台。流编程模型很适合图像处理应用，由于需要传递较大的图像数据，因此这类应用中数据流的特征比较明显。借助于该模型，软件可以充分利用图像处理应用固有的局部性和并行化特点，从底层架构获得较高的性能增益。比如，在MPEG-2编码应用中，期望的处理性能可达到每秒183亿次运算（18.3GOPS），相当于每秒处理105帧，每帧720×480像素、24位表示的彩色图像，功耗为2.2W（Khailany等2001）。

图4-10所示为Imagine流处理器架构的模块。它包括一个128KB的流寄存器堆（Stream Register File，SRF）、48个浮点算术运算单元——分成8个算术运算单元簇且由一个微控制器控制、一个网络接口、一个具有4个SDRAM通道的流存储系统，以及一个流控制器。开发人员设计了一组流指令：Load（加载）和store（存储），前者将数据从片外的SDRAM加载到SRF，后者将来自SRF的数据存储到片外SDRAM，从而实现访问外部数据的目的；Send（发送）和Receive（接收）指令用于发送和接收从SRF到其他Imagine处理器，或者到由外部网络连接的处理单元的数据流；Cluster op（簇操作）指令将流加载到处理单元然后再将返回的流存储到SRF；Load microcode（加载微代码）指令将内核微代码的576位VLIW指令流从SRF加载到微处理器指令存储单元（共2048条指令）（Khailany等2001）。Imagine支持长达32k字的流。处理器由主处理器配置，主处理器发送流指令到流控制器进行译码，然后指示流控制器将其发送到其他片上模块。

Imagine的一个亮点在于处理内核可以执行复合操作；这些内核从SRF的输入流中读取一个元素，然后执行多种算术运算，将结果追加到输出流，再传回SRF。它避免了串行处理器中多种连续的读取和执行指令，由于数据在一个处理阶段可以使用多次，因此显著减少了数据的传输过程。我们可以看到，它和脉动阵列（Kung 1988）通过复用处理器中可用的数据来减少通信带宽的思想相似。

下面通过几个示例来展现Imagine处理器的能力。在7×7的卷积运算中，Khailany等人（Khailany等 2001）演示了Imagine的处理过程：将数据从外部RAM中按行装载像素，然后将这些像素和先前存储的数据（即从SRF加载的累

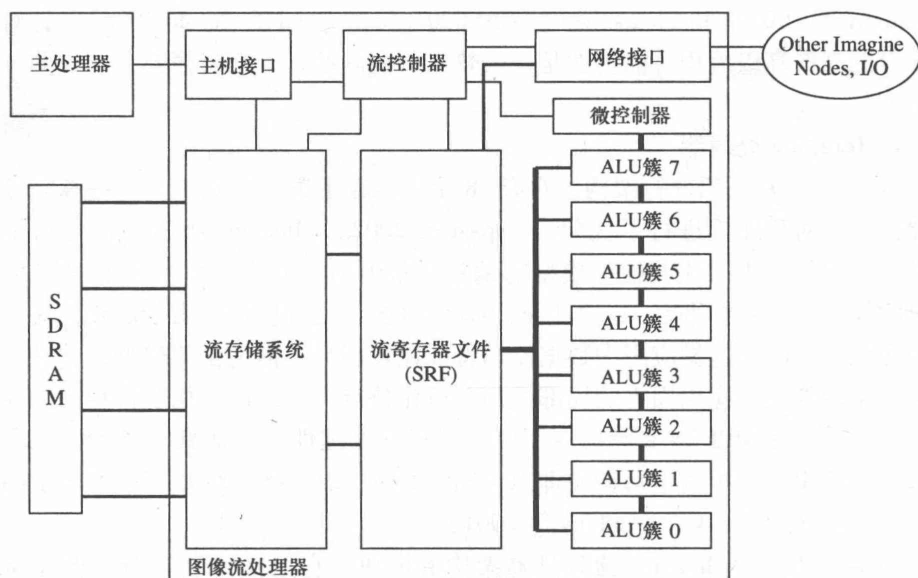


图 4-10 Imagine 流处理器架构（Khailany 等 2001）

积中间乘积）一起分别分发到 8 个处理器。一直执行这些运算直到全部完成。该示例展示了 Imagine 的几个重要特性，包括高效装载数据的能力、处理引擎中复用数据及处理引擎中有效利用并发机制等，使得 Imagine 能够达到较高的性能水平。

由于目前很多应用的核心图像块尺寸都是 8×8 的图像块，并且能够有效地匹配处理需求和计算需求，因此很自然地选择 8 个处理器。对于较小的图像处理操作，比如多种图像处理运算中使用的 3×3 图像块（如 Sobel 和 Laplace 滤波器），无法发挥性能优势。

目前有一种分层的方法用于优化带宽，最高带宽能力预留给处理引擎，达到 544GB/s，SRF 带宽为 32GB/s，到 SDRAM 的片外带宽为 2.67GB/s。在 Imagine 处理器中，采用了很多原本用来获得 FPGA 实现（生成详细的电路结构）的性能增益的优化，但仅局限于部分应用。如第一章所讨论的，从最新的 FPGA 系列器件来看，目前的趋势是开发专门针对部分应用的器件。

2. Storm 流处理器

流处理器的概念由 Stream Processors 公司（SPI）率先在其 Storm 处理器中应用，SPI 是一家从斯坦福研究机构独立出来的无晶圆厂半导体公司。Storm－1 处理器系列被认为对其他 DSP 技术做了很大的改进，见表 4-3。

表 4-3　Storm 流处理器与其他处理器比较（Stream Processors Inc. 2007）

供应商设备	SPI Storm – 1 SPI6HP	Xilinx Virtex™ – 5 5VSX35T	Altera Stratix III EP3SL70	TI TMS320 C6454
构架	DSP	FPGA	FPGA	DSP
时钟频率	700MHz	550MHz	300MHz	1GHz
GMACS/芯片	112	106	86	8
设计方式	软件可编程	可重置硬件	可重置硬件	软件可编程

Storm-1 处理器包括一个处理系统任务的主 CPU（即系统 MIPS）和一个拥有 DSP MIPS 的 DSP 协处理器子系统，后者负责运行向数据并行单元（Data Parallel Unit，DPU）发起核函数调用的主线程。DPU 由一个数据并行单元组成，它含有 16 个或者 8 个数据并行执行通道（依赖于器件的选择），而 Imagine 处理器只有 8 个数据并行执行通道。每个通道或是簇中有 5 个 ALU 单元，另外还有一个指令获取单元和获取 SRF 的 VLIW 序列发生器。DPU 调度器通过接收的核函数调用指令来管理运行期内核及流负荷。与 Imagine 处理器相同的是，一个核函数每次都是跨通道执行，对存储在通道寄存器内的局部流数据进行操作。每个通道有一组 VLIW ALU 和分布式的操作数寄存器堆（Operand Register File，ORF），从而能够处理大量的数据并且提供较大的局部带宽。通道开关是通道间高速存取的全交叉总线，它在编译时被调用，可以认为是 Imagine 处理器的 SRF 基本总线交互的增强。在本书写作时，该处理器的价格为 149 美元/10000 块。

从性能图上可以看到，Storm-1 处理器的性能显著地超越了 TI 公司的处理器的性能，因为它在开发过程中一开始就采用并行化技术，并且通过简单的编程模块来挖掘并行化的能力。然而，在较低的处理速度下，FPGA 的性能也相差不大。表 4-3 再一次强调了利用电路结构的优化来获取性能的重要性。

3. Clearspeed CSX600 处理器

CSX600（Clearspeed Tech. plc 2006）是 Clearspeed 公司开发的嵌入式低功耗并行数据处理的协同处理器。其技术主要针对精细运算，比如矩阵和矢量运算、展开的独立循环和多个并行数据通道，上述运算都可以通过并行处理获得增益，也是经典的 SIMD 计算。目前来看，架构的发展主要面向高性能计算型应用，为基本线性几何子程序（Basic Linear Algebra Subprogram，BLAS）库函数，执行线性几何运算，如矢量和矩阵相乘等子程序的标准应用编程接口，以及数值计算软件库 LAPACK（Linear Algebra Package）等提升了性能。本书列举了几个示例，比如提高 Matlab 仿真速度及在分子动态学力场分析的应用（Amber 9 Sander 隐式方法）。虽然是 C 语言编写，但可以利用不同的库函数将其转化为 C + + 和 Fortran 应用，从而可以与处理器进行交互。

CSX600 能支持每秒 250 亿次单精度或双精度浮点运算，平均功耗约为 10W。其架构采用 64 位寻址，因此通过局部错误检查和纠正（Error Checking and Correcting，ECC）保护的内存接口可以访问数千兆字节的 DDR2 SDRAM，进而提供必要的内存带宽。CSX600 架构的中央是一个处理器引擎，称为多线程阵列处理器（Multi-Threaded Array Processor，MTAP），因此可以在一个核 DSP 函数中做并行计算。另外，该架构采用 ClearConnectTM 的片上网络技术来解决处理器引擎获取或输出数据的带宽需求。

CSX600 的架构包括一个 MTAP 处理器核，该处理核由 96 个高性能的聚合 PE 核组成，每个 PE 核都有 6KB 的专用局部存储器。此外，CSX600 还包含 128KB 的片上便签存储器，一个外部 64 位 DDR2 DRAM 接口，通过 ClearConnect 片上网络连接的 64 位虚拟而物理上是 48 位寻址的缓存，以及多种指令和数据缓存。聚合 PE 的结构如图 4-11 所示。其中一部分 PE 执行一次性的运算，并且执行分支和线程切换等程序控制，而高度并行计算的处理则在处理核的其他部分完成。

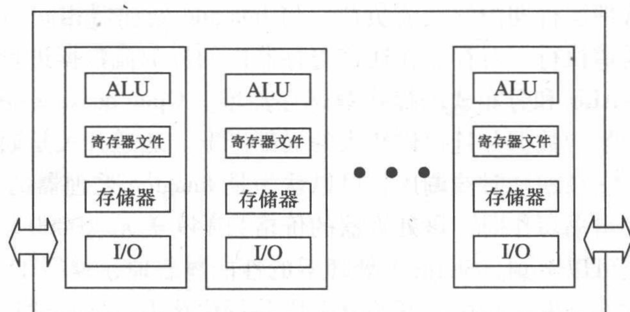


图 4-11 Clearspeed CSX 聚合执行单元

4.7.3 MIMD 架构

MIMD 类型的架构由一组通过多种网络拓扑连接的处理有组成，每个处理器有各自控制单元用来发布指令。对不同 MIMD 架构进行分类的一种主要方法是基于存储器组织方式。其中一种是共享存储器模型，即一组处理器均等地共享存储器空间并且通过读写共享存储器的方式通信。前提条件是所有处理器都有相同的机会访问存储器。另一种是分布式存储器模型，处理器有各自的局部存储器，处理器间通信通过消息传递方式实现（Sima 等 1997）。由于每个处理器都有主通道存储器，因此避免了存储器冲突。此外，这种方式扩展性更好，因为共享存储器架构中的连接网络限制了存储器访问带宽。由于可扩展性方面的限制，共享存储器 MIMD 架构中的处理器数量少于分布式处理器 MIMD 架构中处理器的数量，

后者可扩展到大量并行的计算机。

计算机群可以看作一种分布式存储器 MIMD 架构，其中，大量计算机（PC 机、工作站或者对称式多处理器系统（Symmetric Multiprocessor，SMP）通过高速互连技术，例如 Myrinet（Boden 等 1995）实现网络连接。与大量并行计算机相比，虽然这种架构的连接速度慢很多，但它提供了一种低成本的解决方案。

TILE64™多核系统

TILE64™是最近上市的一款多核处理器，由 64 个相同的处理器核组成，我们称之为“贴片”，通过具有专利权的片上网络拓扑 Mesh™进行连接。TILE64™架构如图 4-12 所示。每一块贴片都是有完整功能的处理器，包括集成的一级（L1）缓存和二级（L2）缓存，以及将各个贴片连接成网状的非阻塞式开关。存储器局部化意味着每一块贴片都可以独立运行各自的操作系统，或者多个贴片联合运行一个多处理操作系统，比如 SMP Linux（Tilera Corp. 2007）。

4.8 专用 ASIC 和 FPGA 解决方案

迄今为止，DSP 技术所能提供的能力都是以预先定义的架构能力展现的。专用 ASIC（大量应用于 FPGA 的实现）主要的吸引力在于其架构可以根据算法需求来开发，使并行化程度与性能需求匹配。例如，前面所讨论的 128 抽头的 FIR 滤波器。在 FPGA 和 ASIC 实现中，有可能分别给每次乘法和加法使用专门的乘法器和加法器，因此能够实现完全的并行运算。并且，为了加速计算，还可以采用流水线方式，在实际中，重复 N 次可以将速度提高 N 倍。本书第 8 章阐述了如何按照性能需求来设计那些必要的硬件。

当考虑可编程能力时，为了验证技术的可行性，ASIC 解决方案不会通过添加额外的硬件来提供可编程能力，是因为性能（速度、面积和功率）可能占主要因素。此外，额外的可编程能力会增加测试与验证次数。非重复性工程（Non-recurrent Enginerring，NRE）成本就是指那些批量生产原型时超过十亿美元的成本。因此，使用专用 ASIC 硬件的观点逐渐具有吸引力。

从本质上来说，FPGA 解决方案通过给予设计者部分可编程空间来避免这些高 NRE 成本。尽管面积、速度，特别是功率性能不具有吸引力，但提供部分可编程或可配置空间的思想避免了 NRE 成本问题，并且显著降低了设计风险——因为留有可配置的空间，所以可以校正设计过程中的错误。

粗略地说，FPGA 可以认为由下面 3 部分组成：

1）可编程逻辑单元，可通过编程实现不同数字运算函数；

2）可编程连接，连接不同模块；

3）可编程 I/O 引脚。

借助于这种可编程能力，开发者可以修改 FPGA 上已经实现的功能，或者通过不同的方式连接 FPGA 上已有的功能模块，甚至通过不同的方式连接不同 FPGA 上的电路。在早期，改变相互连接的能力是主要的吸引力，但由于 FPGA 变得越来越复杂，因此这种能力仅仅作为改变实际功能的补充。从高效设计的角度考虑，主要目标是以最有效的方式利用大量的处理能力，从而能够在性能上超越其他技术。比如 ASIC，它要求设计合适的电路结构来充分利用底层硬件资源。长期以来，它都被当成是一种硬件设计过程，因为与软件设计相比，需要投入更多的时间和精力。然而，本书后面将讨论的一些示例所展现的性能优势非常明显，需要在设计过程中考虑，这也是本书的重点。

4.9 总结

本章重点讲述了实现 DSP 复杂系统的多种不同技术，从速度、功耗和面积上进行了比较，虽然面积的比较对于处理器实现来说难以确定。本章还对这些技术的可编程能力进行了探讨，着重强调了底层芯片架构会限制性能。事实上，可以为 ASIC 和 FPGA 技术设计专用电路架构，从而可以获得较高的性能。我们甚至可以认为，FPGA 可更改的电路架构及其可编程能力是 FPGA 技术在某些系统实现中具有吸引力的两大因素。

尽管本章主要描述不同的技术并进行了对比，但实际中，现代 DSP 系统常常将这些不同平台综合到一起。目前，很多公司提供的复杂 DSP 板同时包含微处理器、DSP 微处理器和 FPGA，有的公司，比如 IBM 提供嵌入式 FPGA 设备。由于不同平台适合不同的计算需求，因此这种实现方式也不足为怪。当前的 DSP 微处理器及下一章将介绍的最新 FPGA 技术可以看作是包括多个硬件组件的异构平台。

参考文献

Analog Devices Inc. (2000) Adsp-2106x sharcfi dsp microcomputer family: Adsp-21060/adsp-21060l. Web publication downloadable from http://www.analog.com/.

Barnes GH, Brown RM, Kato M, Kuck DJ, Slotnick DL and Stokes RA (1968) The ILLIAC IV computer. *IEEE Trans. Computers* C–17, 746–757.

Berkeley Design Technology (2000) Choosing a DSP processor. Web publication downloadable from http://www.bdti.com.

Boden NJ, Cohen D, Felderman RE, Kulawik A, Seitz C, Seizovic J and Su WK (1995) Myrinet: A gigabit-per-second local area network. *IEEE Micro* pp. 29–36.

Clearspeed Tech. plc (2006) Csx600 datasheet. Web publication downloadable from http://www.clearspeed.com.

Constantinides G, Cheung PYK and Luk W 2004 *Synthesis and Optimization of DSP Algorithms*. Kluwer, Dordrecht.

Dahnoun N (2000) *Digital Signal Processing implementation using the TMS320C6000^{TM} DSP Platform*. Pearson Education, New York.

Hennessey J and Patterson D (1996) *Computer Architecture: a quantitative approach*. Morgan Kaufmann, New York.

Hillis W (1985) *The Connection Machine*. MIT Press, Cambridge, MA, USA.

Inmos (1989) *The Transputer Databook (2nd edn)*. INMOS Limited. INMOS document number: 72 TRN 203 01.

Intel Corp. (2001) Ia32 intel architecture software developers manual volume 1: Basic architecture.

Jackson LB (1970) Roundoff noise analysis for fixed-point digital filters realized in cascade or parallel form. *IEEE Trans. Audio Electroacoustics* AU–18, 107–122.

Kapasi U, Dally W, Rixner S, J.D. O and Khailany B (2002) Virtex5.pdf *Proc. 2002 IEEE Int. Conf. on Computer Design: VLSI in Computers and Processors*, pp. 282–288.

Khailany B, Dally WJ, Kapasi UJ, Mattson P, Namkoong J, Owens JD, Chang A and Rixner S (2001) Imagine: Media processing with streams. *IEEE Micro* 21, 35–46.

Kung HT and Leiserson CE (1979) Systolic arrays (for VLSI) *Sparse Matrix proc. 1978*, pp. 256–282.

Kung SY (1988) *VLSI Array Processors*. Prentice Hall, Englewood Cliffs, NJ.

McCanny JV and McWhirter JG (1987) Some systolic array developments in the UK. *IEEE Computer Special issue on Systolic Arrays*, pp. 51–63.

Sima D, Fountain T and Kacsuk P (1997) *Advanced Computer Architectures:A Design Space Approach*. Addison-Wesley, Harlow, UK.

Stream Processors Inc. (2007) Stream processing: Enabling a new class of easy to use, high-performance DSPs. Web publication downloadable from http://www.streamprocessors.com.

Texas Instruments Inc. (1998) Tms320 DSP development support reference guide. Web publication downloadable from http://www.ti.com/.

Tilera Corp. (2007) Tile64 product brief. Web publication downloadable from www.tilera.com.

Woods R and Masud S (1998) Chip design for high performance DSP. *IEE Elect. and Comms. Jour.* 10, 191–200.

Woods RF, McCanny JV and McWhirter JG (2008) From bit level systolic arrays to HDTV processor chips. *Journal of VLSI Signal Proc.* Special issue on 20 years of ASAP. DOJ 10.1007/S11265-007-0132-z. ISSN 0 922-5773.

第5章 当前的 FPGA 技术

5.1 引言

通过在前面章节对不同技术的详细介绍，我们对具体技术的选择变得清晰了，而且这些技术所包含的设计方法显示了所述的专用 DSP 系统能达到的性能等级。比如，使用简单 DSP 微控制器表明这是一个性能要求相对低的 DSP 系统，也意味着用户需要为此设计 C 或者 C + + 代码。但是，用户还可以使用 Matlab® 或 Labview 来满足诸如系统需要的字长或数量等设计环境要求，也可以通过所述设计方法与软件例程来产生微控制器的 DSP 源码。当然有人会提出通过这种方法产生代码的效率不高，但是这种方法足以满足应用的性能要求，同时仍然使用较实用的低功耗微控制器，这也减少了设计时间，满足了成本要求。

这个设计方法适用于所有的“处理器”类型平台，但是可能为达到必要的性能，需要人工开发专用 C 代码。这在一些应用中很必要，如性能要求严格的应用（而且不能通过提高平台的计算复杂性来满足），或者具体结构的专用功能不能在高级工具环境里很好支持的应用。这对较新的可重配置或者专用处理器尤为典型，比如 Storm Stream 处理器。很明显，在这些情况下选择该平台是因为它在面积 - 速度 - 功率方面有着优越的性能。平台的吸引力表现在具体特征方面，比如一些商业平台的多重 MAC 单元，或者专用 DSP 平台的专有处理功能，又比如 Storm Stream 处理器的数据并行单元。在这些情况下，为了利用技术的具体架构特色，用户就必须在设计方便上做出让步。

在 SoC 的概念下，用户通过创造电路架构来最大限度满足具体系统的性能要求这一想法是很极端的。虽然用户能创造出最终满足 DSP 系统要求的电路架构，但是这一想法受到创建最终架构实际情况的限制。因而，包含具体现有架构的设计方式或利用系列现有构建模块的设计方式开始成为主导。这体现了一种 SoC 设计风格，即利用有限的功能在合理时间内设计出一个系统。

正如 4.8 节介绍的，这是由一个 FPGA 平台实际提供的。FPGA 已经不再是一个“胶合逻辑”平台，而成为了一个能帮助用户创造 DSP 系统组件集合的平台。本章将详细讲解如何使用 FPGA 技术制作 DSP 系统。购买产品的全部细节可以在各生产厂商的设计说明文档中查到，比如 Xilinx，Altera，Lattice 和 Atmel，本章只强调了其中的重要特征，突出讲解一些对于 DSP 实现很重要的方面。同

时每个生产厂商都有一系列使用不同技术的产品，本章重点讲解来自 Altera 的 Straix® Ⅲ和来自 Xilinx 的 Virtex™ - 5FPGA 系列等最新的能买到的产品。此外，还将介绍不同于其他产品的一些特有技术，比如来自 Actel 的基于闪存技术的 Pro ASIC^{PLUS}FPGA 技术和来自 Lattice 的由 CPLD 概念（相比基于 LUT 方式）扩展的 ispXPLD™5000MX 系列技术。

本章从 5.2 节中 FPGA 的简要历史角度开始，描述它们怎样从一个细颗粒化技术发展为一个有着复杂系统模块的技术。5.3 节将介绍 Altera 的 FPGA 系列，特别讲解 Stratix® Ⅲ FPGA 系列，因为它是该公司最强大的 FPGA 系列的代表。对 MAX® 7000 FPGA 也做了简单介绍，因为它代表了 PLD 概念的革新，Altera 最初的可编程硬件产品正是基于这种架构。5.4 节将接着介绍来自 Xilinx 的 FPGA 技术产品，特别是Virtex™FPGA 系列。这次，我们主要讲解这一技术最新的 FP-GA 版本，称之为Virtex™-5FPGA 系列。与 Altera 和 Xilinx 一起，我们重点关注与 DSP 系统密切相关的硬件方面，但也尽力照顾到 I/O 速度、定时策略和存储器结构的其他方面，因为它们也是决定系统整体性能的重要方面。

Actel，Atmel 和 Lattice 也提出过许多其他技术，而且它们的具体特征与某些市场需求非常相关。比如，Lattice 提出的采用 E^2PROM 非易失性的单元组合来存储设备配置的 ispXPLD™5000MX 系列及采用 SRAM 技术来完成逻辑实现；这些将在 5.5 节介绍。Actel®提出了许多基于闪存和反熔丝技术的 FPGA 技术。最初的 Actel 凭借一种反熔丝技术出名，而且它现在仍然使用这种技术，也就是反熔丝 SX FPGA 技术，这将在 5.6.1 节中介绍。5.6.2 节将介绍 Pro ASIC^{PLUS}闪存 FPGA 技术。这种方法允许设备储存它的程序，避免了像基于 SRAM 的技术中对编程设备的需要。还将介绍该公司最新提出的代表第一个复合信号 FPGA 的 Fu-sion™ 技术。最后将介绍 Ateml®的相对古老的 AT40K FPGA 技术，它提出了部分可重构解决方案。5.8 节将对本章进行小结。

5.2　FPGA 的发展

在 20 世纪 70 年代，逻辑系统是通过由 TTL 逻辑芯片组成的 PCB 板来创建的。但是，随着功能的变强，逻辑规模的增长，更重要的是，逻辑电平的数量的增加，设计速度减慢了，从而限制了系统的发展。最典型的例子是，设计者使用基于 Karnaugh 映射或 Quine- McCluskey 最小化等逻辑最小化的技术来创建基于多个与门和一个对它们求和的或门创建乘积模块等各种产品。

1978 年，Monolithic Memories 提出制作一个架构来实现这个功能，并利用可编程阵列逻辑（Programmable Array Logic，PAL）设备做了验证。PAL 包括一个固定或矩阵及与其连接的一个可编程与矩阵，它可以直接实现最小化表达式的乘

积和结构。与和或矩阵概念成为了可编程逻辑设备的关键特色，表 5-1 给出了一个简单的分类。如图 5-1 所示，固定的与平面（有效的一种译码）和可编程或平面与只读存储器（Read Only Memory，ROM）拥有同样的结构。在这种情况下，这种结构可以被认为具有存储 4 个（通常表示为 2^n）2-（或者 m-）bit 的容量，如图 5-2 所示。地址输入译码器用来减少连接存储器的引脚数量，存储区或存储阵列用于存储数据。译码器使用与门生成不同的地址线，然后使用或门实现输出求和，这实现了布尔运算的与或结构。通常，一个 2^n bit ROM 能实现任何 n 输入的布尔运算，因此一个 4 输入 ROM 或 LUT 成为了最早 FPGA 的核心组件，即 Xilinx XC2000 FPGA。4 输入 LUT 在实现芯片面积的有效利用方面又足够小，在实现一系列合理功能方面又足够大。在需要大量输入时，可以通过串接或并接 LUT 输入实现扩容。这会使实现速度更慢，不过，比起使用更大规模的 LUT，这个方案利用率更高。

表 5-1 PLD 类型

	与矩阵	或矩阵
ROM	固定的	可编程的
PLA	可编程的	可编程的
PAL	可编程的	固定的

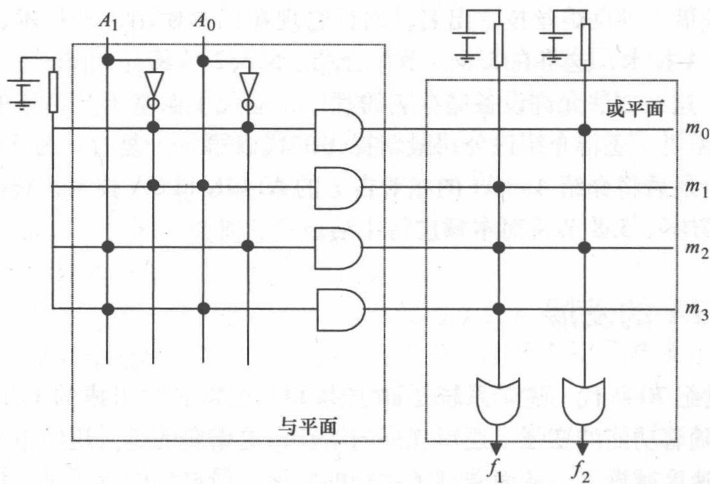


图 5-1 ROM 详细结构

PLD 结构有许多的优点。它明显适用于通过逻辑最小化技术创建乘积和的过程。这一功能可以被加载到 PLD 设备中，如果没有足够的可用乘法单元，则问题会被反馈到第二个 PLD 阶段。另一个主要的优点是电路延时是确定的，包含

一级逻辑电平或两级等。但是，真正的优点是可编程性，这降低了使用硬件 PCB 开发的危险性，这个优点在于允许通过调节 PLD 的逻辑实现来修正可能的错误。但是，随着集成度的增长，把 PLD 作为结构单元的想法成为了一个具有吸引力的 FPGA 建议，这在早先的 Altera 产品中有体现，在目前的 MAX 7000 设备系列中也有体现。前边已经提及，Xilinx 的优势在于 ROM 或者 LUT 方式。

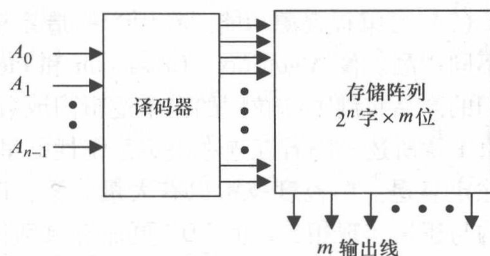


图 5-2　ROM 框图

5.2.1　FPGA 的早期结构

FPGA 的早期产品包含了一个 Manhattan 类型的结构，该结构包含了简单的逻辑结构单元，每个单元都通过可编程的连接连在一起。因此 FPGA 具有如下特点：

1）可以通过编程来实现不同数值函数的可编程逻辑单元；

2）可编程的连接允许不同模块互联；

3）可编程的 I/O 引脚。

FPGA 经历了一个理想的演进过程，最初 FPGA 被看成是简单的胶合逻辑，之后成为提供冗余、防止 PCB 电路板制作错误的关键技术，并且 FPGA 组件可以实现可编程互连，甚至具有纠正不正确系统设计带来的错误的机制。但是，摩尔定律描述的技术演进过程使 FPGA 供应商面临扩展性问题。在 20 世纪 80 年代，FPGA 供应商在可编程模块的数量、互连的阶段数及 I/O 数量方面进行了浓缩扩展。但是，这个方法有些局限，因为浓缩使互连成为了主要问题。同时，随着技术革新，另一种浓缩方法成为可能，即专用硬件单元也可以被集成进去，比如专用乘法器及最新的处理器。此外，系统网络连接的问题可以通过集成 SERDES 和 Rapid I/O 形式的专用网络连接来缓减。

技术革新使 FPGA 技术内涵更加丰富。

（1）技术争论。关于哪个技术更有效率的争论通过摩尔定律得以平息。早期出现了三种不同的 FPGA 技术，即传统的 SRAM、反熔丝和 EPROM 或E^2PROM 技术。后两种技术都需要特殊的工艺步骤，要么需要制作反熔丝链接，要么需要特制晶体管实现 EPROM 或 E^2PROM。技术进步过程选择了 SRAM 技术，因为制造开始标准化，这对 Altera 和 Xilinx 非常重要，因为 FPGA 的制造可以外包，也不必与制造公司做具体专业技术交流。的确，值得注意的是硅制造商现在将 FPGA技术作为检测它们制造设备的最先进技术。

（2）可编程资源功能。在用于构造系统的基础逻辑模块资源方面，存在大量不同产品。像 Algotrinix，Crosspoint 和 Plessey 公司都已经提出了自己的 FPGA，它们的产品布满了细粒度的简单逻辑门或数据选择器，提供了大量逻辑资源。正如第 1 章所述，随着互连在决定系统性能里扮演的角色越来越重要，这些设备就已注定衰退。粗粒度技术也有大量方案，即 PLD 类型的结构或 LUT 结构。PLD 结构与逻辑实现相关，而 LUT 更加有灵活性并且是程序员和工程师已经能够理解的概念。纵观目前的 FPGA 产品，很明显可以看出基于 LUT 的结构主宰了技术的革新，Xilinx Virtex™-5 技术中 LUT 的大小从 4 输入增长为 5/6 输入，而 Altera Stratix® Ⅲ系列已经增长到了 6 输入。

（3）FPGA 市场的变化。随着 FPGA 复杂度的增加，FPGA 已经从胶合逻辑组件发展为复杂系统的主要组件，与 DSP 一起成为了这本书的讲述对象。同时，人们应该能够注意到 FPGA 是电信行业一个重要的组成部分。这意味着 FPGA 供应商必须在与类似产品竞争方面具有优势，首先面对的是 DSP 处理器开发商，比如 TI 公司、Analog 和多核开发商。前面的章节已经介绍了这些技术的某些方面。

（4）工具流。最初，FPGA 并不复杂，直到 20 世纪 90 年代中期，很平常的设计师就能完成手动设计布置。第一个主要的开发工具是自动化布局和路由工具，它仍然在供应商的工具流中扮演一个主要的角色。但是，需要系统级设计工具来处理最新的设计挑战这一理念逐渐被公认。因此，FPGA 供应商已经开始系统级设计工具的研发，除了来自工具供应商的研发的系统级设计工具，还有 Altera 的 DSP 和 SOPC 生成器，以及 Xilinx 的 DSP 和 AccelDSP™的系统生成器。这是一个日益突出的问题，因为工具有滞后于技术发展的趋势，也是这本书中的一个主题。

现阶段，FPGA 代表着系统平台。从两个主要的供应商可以看到，他们现在用这些术语来介绍他们的技术：Xilinx 描述他们的Virtex™为一个 FPGA 平台；Altera 也一样，认为他们的高端产品 Stratix Ⅲ能够设计整个 SoC。因此，我们已经从本节开始突出讲述的具有由可编程连接连起来的可编程单元的时代，转移到了复杂的、可编程的 SoC 设备，它包含了大量的所谓专用 DSP 处理器模块、软处理器引擎及硬处理器引擎等关键组件。

5.3 Altera 的 FPGA 技术

Altera 是两个主要的 FPGA 公司之一，而且他们最初的基于 PLD 结构的架构在不断升级，这在前面的章节中介绍过了。它目前的 FPGA 系列被系统化到了几个不同的技术中，见表 5-2。Altera 的 FPL 系列被系统化到 MAX® 和 MAX® Ⅱ系

列的可配置可编程逻辑器件（Configurable Programmable Logic Device，CPLD）；Cyclone 和 Cyclone Ⅱ系列的低功率 FPGA；Stratix®、Stratix® Ⅱ、Stratix® Ⅲ和 Stratix® GX 系列的高密度 FPGA；结构化的 ASIC 解决方法 HardCopy®和 HardCopy® Ⅱ系列。本节将主要集中于讲述 Stratix® Ⅲ系列，因为我们主要讲述 DSP 及超高性能的 DSP 应用，但是对 MAX 7000 系列 FPGA 也会简单回顾，因为它是 PLD 概念的典型扩展。

表5-2 Altera 的 FPGA 系列范围

类型	系列	简单介绍
CPLD	MAX® Ⅱ	有着许多有联系的基于 PLD 模块的技术，其中的系列包括 MAX® Ⅱ、MAX® 3000A 和 MAX® 7000
FPGA	Cyclone	成本最佳化，内存充足的 FPGA 系列
FPGA	Stratix®	高性能、低功耗的 FPGA
FPGA	Stratix® GX	有着高速的连续收发器的 FPGA，这种收发器有着可扩展、高性能的逻辑阵列
结构化的 ASIC	HardCopy®	低功耗、高性能的结构化 ASIC，其中有着实现 Stratix Ⅱ设备的输出引脚、密度和架构

Altera FPGA 的核心模块已经成为了逻辑元件（Logic Element，LE），如图5-3所示。在 XC4000 和早期 Virtex™ FPGA 系列中可以看到，这与 Xilinx 的逻辑单元（Logic Cell，LC）非常相似，虽然 Xilinx 最近已经转移到了一个6输入的 LUT。这个单元是从一个概念建立起来的，它的目的是要实现标准化的组合逻辑功能（仅 LUT）、延时或移位功能（仅触发器），或者时序逻辑电路（注入触发器的组合逻辑）。因此所有配置、连同各种多路技术和互连网络是一起提供的。4输入 LUT 的概念出现在 Rose 等（1990）中，里面显示了该规模的 LUT 为大量不同的实例生产最好的面积效率。使用这些4输入 LUT，通过使用可编程互连来连接这些4输入的 LUT，从而建立更大的 LUT，最终实现各种组合逻辑。正是基于这种方法，我们为加法器的实现建立了一个快速进位的逻辑电路，如图3-3b 所示。

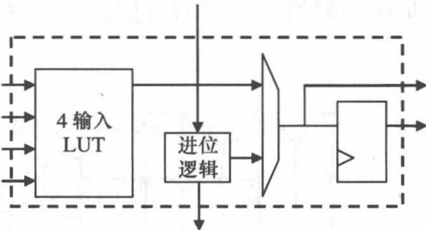


图5-3 Altera LE 单元的框图

5.3.1 MAX® 7000 FPGA 技术

图5-4所示为基本的 PLD 结构，图5-5中详细给出每个 PLD 模块所具有的形式。这个框架将为 PLD 模块经可编程互连电路连接在一起提供解决方案，随

着优势不断展现，PLD 的重要性已经在更大范围凸显出来了，通过使用商业 PLD，可以把几个 PLD 芯片连接起来实现更复杂的逻辑模块，而优势更明显的是基于 PLD 的 FPGA 可编程互连体系。

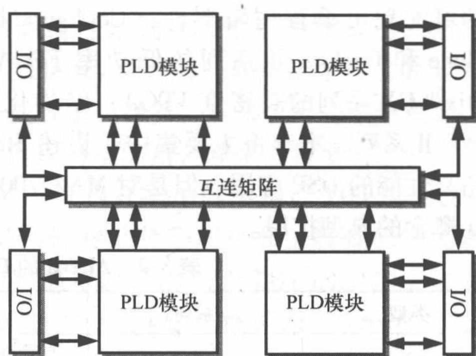


图 5-4 普遍的基于 PLD 的 FPGA 的架构

来自 Altera（Altera ⓐ Corp. 2000）的 MAX® 和 MAX® Ⅱ 系列都是 PLD 技术基本概念的扩展。图 5-6 中所示为 MAX7000 的架构，该架构包含了 16 个宏单元逻辑阵列模块（Logic Array Block，LAB）、一个允许模块间相互连接及时钟、复位等各种各样的控制输入连接的可编程互连阵列（Programmable Interconnect Array，PIA）、允许其内部接口连接到 LAB 和 PIA 的 I/O 的控制模块。4 个专用输入允许，比如时钟、复位和使能等高速的全局控制信号注入到宏单元和 I/O 引脚中。每个 LAB 经由来自 PIA 的 36 个一般逻辑信号直接反馈、用于部分功能寄存器的全局控制，以及从 I/O 引脚到寄存器的直接连接，这些都是为了片外和片上的通信延迟最小化。

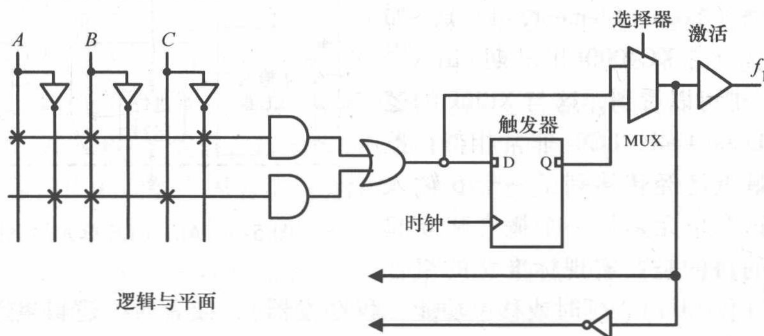


图 5-5 PLD 构造模块

图 5-7 所示的 MAX7000 宏单元是关键的计算部分，它用于时序逻辑运算或组合逻辑运算。它包含了一个逻辑阵列、一个乘积项选择矩阵和一个可编程寄存器。这个逻辑矩阵允许 5 个乘积项求和，而且乘积项选择矩阵允许作为每个宏单元中一个单元的主要输出来使用，因此能实现组合输出，也可作为一个更大的逻辑功能组成部分，即到寄存器的一个第二输入，也可以用来实现时序逻辑电路。触发器可用于延时，或者作为一个更大的时序逻辑电路的一部分，也可以使用各种各样的选择功能通过复位或时钟实现多种控制。这个单元也包含了可共享的扩

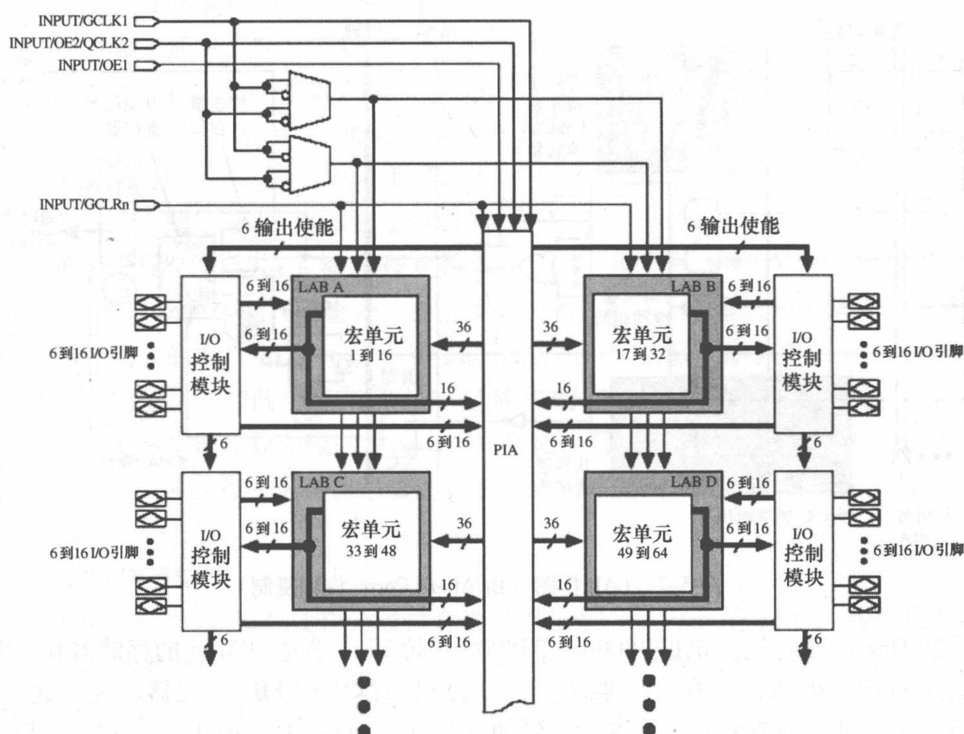


图5-6 Altera MAX7000E 和 MAX7000S 设备框图（由 Altera Corp. 许可复制）

展器和并行扩展器，可共享的扩展器允许乘积反转项馈入到逻辑阵列，而并行扩展器允许使用来自宏单元的输入创建更大的扇入逻辑功能。

逻辑信号可以通过 PIA 在 LAB 之间传输，它形成一个可编程的路由，从而把任何信号传输到设备上的任何地方。该路由是通过对一个 E^2PROM 单元编程来实现的，它通过控制一个 2 输入与门的其中一个输入实现断开或使能连接。PLD 的一个关键优势成为 FPGA 选择 PLD 的重要理由，即路由延迟是固定的，不像后面讲到的其他 FPGA 架构，它们的延迟是累加的、变化的，而且有着路径依赖性，并且会在时序闭合（时序闭合是指设计中所有单独的延迟必须满足各种时间约束，即关键路径和边到边时延的约束）时造成问题。因此这使时延容易预测。从 DSP 角度看，这些设备的使用有限，是因为主要组合模块只能用于传统逻辑实现。

5.3.2 Stratix® Ⅲ FPGA 系列

Stratix® Ⅲ FPGA 系列做了许多的改变。针对 DSP 中以数据为中心的应用，Stratix Ⅲ E 系列上包含了充足的存储器和乘法器。Stratix Ⅲ E 系列具有许多很有

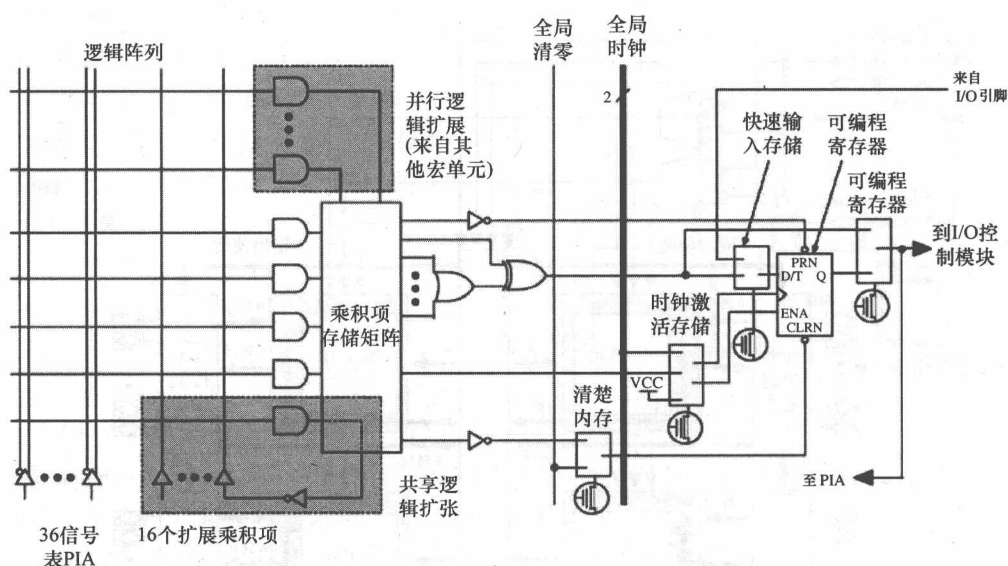


图 5-7 LAB 框图（由 Altera Corp. 许可复制）

用的 DSP 应用特征，包括 48000 ~ 338000 等价 LE，多达 20Mbit 的存储器和许多高速的 DSP 模块，具有专用乘法器、复合积加模块和 FIR 滤波器功能。此外，设备也提供可调节的电压等级、大量的锁相环（Phase Locked Loop，PLL）和各种时钟。图 5-8 所示为 Altera Stratix EP3SE50 的平面图。

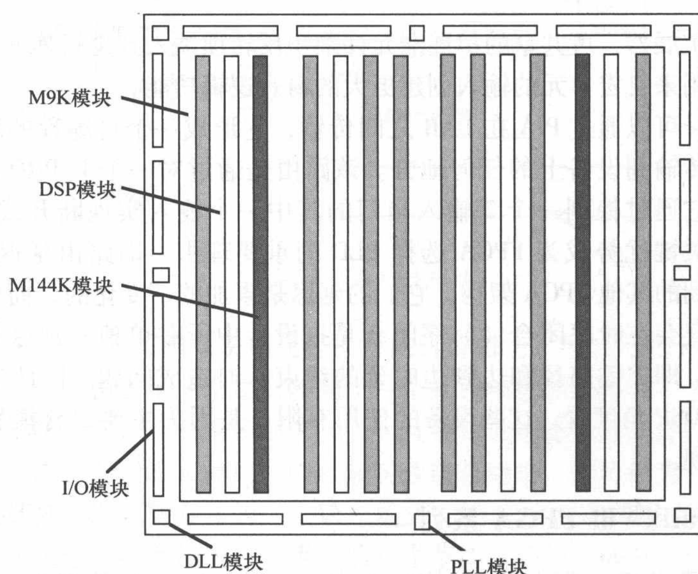


图 5-8 Altera Stratix EP3SE50 平面图

1. 自适应逻辑模块

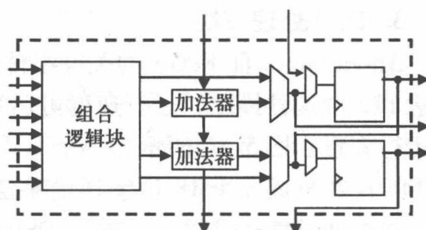


图5-9 Altera ALM框图

在Stratix Ⅲ中，扩展了LE概念，Altera称之为一个自适应逻辑模块（Adaptive Logic Module，ALM），如图5-9所示。ALM基于可分裂的8输入LUT，允许图5-3所示的原始LE配置；也允许很多其他组合输入，包括7输入和6输入的组合，5输入和3输入的组合，甚至5输入和5输入、4输入和5输入，以及6输入和6输入的LUT组合（只要单独输入的总和不超过8就可以）。此外，还有两个专用的加法器和两个寄存器。在ALM中的寄存器与LUT的比例为2:1，以确保FPGA不是寄存器受限的。两个加法器能进行2位加法或一个单独的三进制加法。LUT－乘法器－寄存器组合的核心概念仍然保留，不过这将变成一个更大的LUT。

2. 存储器结构

Altera Stratix FPGA具有分级储存器体系，即TriMatrix存储器，具有从小的分布式存储器模块到大的17Mbit容量的存储器模块的分布范围，运行速率超过600MHz。表5-3中列出了存储器。包括的三种类型。

MLAB模块或储存LAB是LAB的衍生物。MLAB是LAB的一个父集而且能实现640位的简单双端口SRAM。因为每个ALM能够被配置成64×1或32×2模块，所以MLAB能够被配置成像一个64×10bit或32×20bit的简单双端口SRAM模块。MLAB和LAB模块成对共存，这允许50%的LAB被换成MLAB。DSP应用中，MLAB被用于本地存储器来储存临时的本地数据，因为大量的存储器能并行存取，所以性能很高。

M9K是9KB模块RAM，允许存储大量数据，如图5-8所示。

比M144K模块更大的是144KB RAM，在DSP的图像处理应用中，可以用于储存图像。

每个嵌入式存储器模块能独立配置成单端或双端RAM，或者移位寄存器。

表5-3 Stratix存储器类型及用法

存储器类型	位/模块	模块的编号	建议的用法
MLAB	640	6750	移位寄存器，小的先入先出缓存，滤波器先入先出缓存区，滤波器延迟线
M9K	9216	1144	一般目的的应用，打包头文件，单元缓存区
M144K	147456	48	处理器编程存储，打包或录制帧缓冲器

3. DSP 处理模块

Altera StratixⅢ FPGA 的关键组件是 DSP 功能模块。最大的 Stratix 设备支持多达 112 个这种模块，运行速度可达 500MHz，从而为大量的 DSP 应用提供巨大的运行容量。图 5-10 所示为半个 DSP 模块的详细框图。输入寄存器模块的字长是 144 位，分成 8 个 18 位字送给乘法器，如 dataa 和 datab 所示，输出是 72 位的。数据通过寄存器输入、输出 DSP 模块，从而避免了用 DSP 模块构成一个更大系统时关键路径的延时问题。除了寄存器输入，为了更高的速率，DSP 模块具有可选的流水线技术，如图中流水线寄存器库所示。在后面的章节会看到，在电路架构中必须考虑实现 DSP 功能的流水线延时。

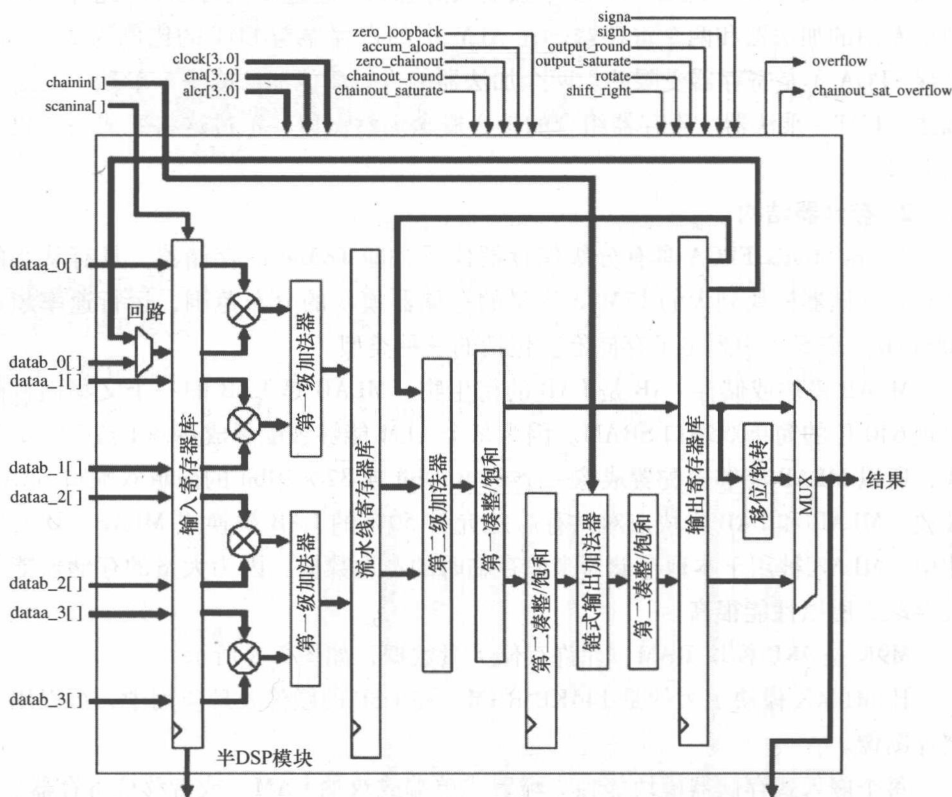


图 5-10 Altera Stratix DSP 模块框图（Altera Crop. 2007）（由 Altera Corp. 许可复制）

模块第一级的两个双向的乘法/累加模块已经升级，可以支持特殊的 DSP 功能，也就是对每个模块的 2 抽头 FIR 滤波器做了配置，在选择的流水线之后利用第二级加法器/累加器对两个 2 抽头滤波器求和，从而形成一个 4 抽头滤波器；FFT 部分蝶形运算型阶段；以及其他的复杂运算，如复数乘法，即在输入级完美实现了 $a+\mathrm{j}b$ 乘上 $c+\mathrm{j}d$ 等于 $(ac-bd)+\mathrm{j}(ad+cb)$ 的复杂运算。乘法器是 18

位的，也能实现 9 位乘法。Altera 的文献中（Altera Corp. 2007）指明 9 位、12 位、18 位及 36 位字长同时支持有符号和无符号。

复合累加阶段之后是可选择流水线寄存器库，再之后是第二级加法器/累加器。再次利用了加法器优势来实现快速的 DSP 系统，通过输出寄存器库的反馈重现功能实现计算，这一般能在 IIR 滤波器实现中看到。此外，DSP 模块可以利用链式输出加法器将以上 DSP 模块的输出相加；这是因为 DSP 模块在纵列中是相连的，如图 5-8 所示。这表明用这种方法连接两个 DSP 模块可以实现 4 抽头滤波器，也可以实现更大的滤波器，只是需要更多的 DSP 模块。

凑整与饱和模块被放置在第二级加法器/累加器和链式输出加法器模块之后。第一级里的字长是可以预见的，但是由于输出经回环不断地回馈，使用凑整或饱和模块来避免溢出很有必要。同样，当创建大型滤波器时，该模块也适用于链式输出加法器。第 4 章已经讲了其中的原因。

总之，DSP 模块能够完成 5 个基础的 DSP 运算，见表 5-4，它是对数据手册内容的汇总。这里给出了算法（有符号/无符号）与能否凑整的更多细节。支持两种凑整模式，即 DSP 中常用的凑整方法：四舍五入模式，向偶数凑整模式。也支持两种饱和模式，即对称和非对称饱和。在二补数形式中，最大的负数表示为 -2^{n-1}，例如 8 位时为 -128，最大的正数是 $2^{n-1}-1$，例如 8 位时为 127。在非对称情况下，这是任何数字的饱和范围，即从 $-2^{n}+1$，例如 -127，到 $2^{n-1}-1$，例如 127。对于在 44 位表示法中凑整和饱和有 16 种不同形式，需要根据应用允许准确度动态范围调整。

表 5-4　DSP 模块运行的模式（Altera Corp. 2007）

模式	乘数宽度	乘法器数量	每个模块
独立乘法器 9	1	8	
	12	1	6
	18	1	4
	36	1	2
	双倍	1	2
双乘法器的加法器	18	2	4
4 乘法器的加法器	18	4	2
复合累加器	18	4	2
移位	36	1	2

4. Stratix 时钟网络和 PLL

在 FPGA 技术中，有时深入理解 DSP 模块很重要，但看一下定时策略也很重要，因为它是达到需求性能的关键。Stratix Ⅲ 设备有大量的专用全局时钟

(Global Clock，GCLK) 网络、局部时钟 (Regional Clock，RCLK) 网络及外围时钟 (Periphery Clock，PCLK) 网络。这些被统一纳入了一个分层次的时钟结构，它提供了高达 220 个特别的时钟域 (16GCLK + 88RCLK + 116PCLK)。因为时钟网络能耗巨大，所以 Quartus® Ⅱ 软件编译器会自动关闭设计中不用的时钟网络。Stratix 每个设备具有高达 12 个 PLL 而且每个 PLL 有高达 10 个输出，他们每一个都能被独立编程，可定制时钟频率，并且与其他输入或者输出时钟没有固定的联系。

总共提供 16 个 GCLK，它们被纳入了一个 H 时钟树网络，如图 5-11 所示。这为每个时钟信号提供一个相等的延时，从而消除时钟偏移。GCLK 信号可以驱动功能模块，如 ALM、DSP 模块、TriMatrix 存储器模块及 PLL。Stratix Ⅲ设备 I/O 元件 (I/O Element，IOE) 和内部逻辑也能驱动 GCLK 来在内部生成全局时钟和其他高扇出控制信号，如全局复位或时钟使能。RCLK 网络仅占驱动的四分之一，且能作为全局复位或使能时钟来使用。PCLK 网络是来自 Stratix Ⅲ设备外围驱动时钟网络的集合。

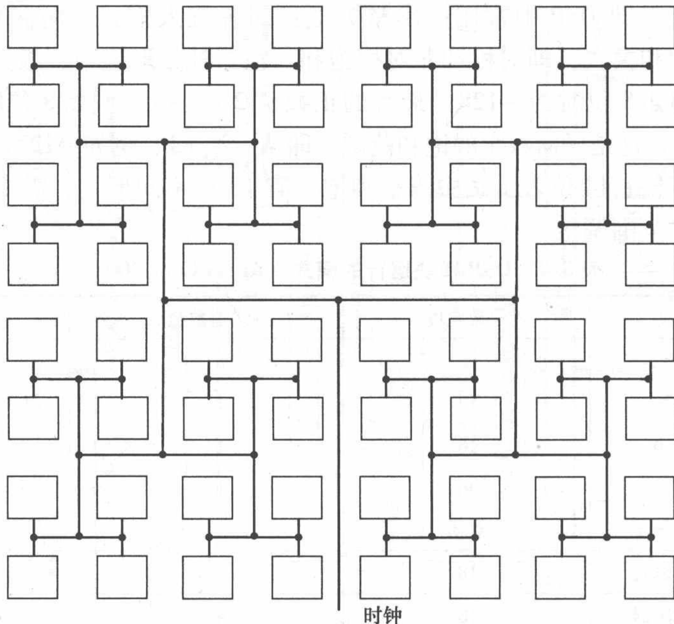


图 5-11 H 时钟树缓存区

另外，PLL 能实现内部或外部时钟 f_{co} 与一个输入参考时钟 f_{ci} 的相位同步或频率同步。图 5-12 所示为在 Altera Stratix 中使用的一类 PLL。电压控制振荡器 (Voltage Controlled Oscillator，VCO) 产生一个周期输出信号。假定振荡器产生一个接近的频率作为参考信号。VCO 输出时钟频率与参考时钟 f_{ci} 的相位差可以通

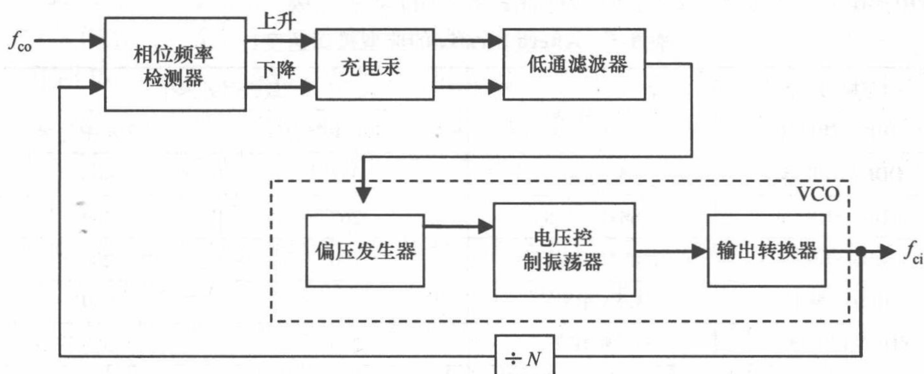


图5-12　相位锁定回路

过一个叫作相位频率检测（Phase Frequency Detector，PFD）的模块检测到。然后产生有效的控制信号来控制VCO频率的上升或下降。这些信号通过充电汞的电路来改变控制电压，最终控制VCO速度上升或速度下降。低通滤波器也叫回路滤波器，用来使充电汞陡峭的控制输入变得平滑，从而防止电压过冲并且将这些上升或下降的信号转变为一个电压，再经偏压发生器来偏置VCO，由此决定VCO的运行速度。反馈环路中的除法器用于提高VCO的频率，使输出频率高于输入参考频率f_{co}，这就意味着VCO到PFD的反馈时钟是输入参考时钟f_{co}的N倍。因此，输入PFD的反馈时钟被锁在时钟f_{ci}上，f_{ci}又可以作为另一个PLL的参考信号。

在Altera Stratix FPGA中，FPGA的左边/右边PLL的VCO输出7个缩放后的计数器，同时顶部/底部PLL的VCO输出10个缩放后的计数器。这些计数器通过PLL可以产生许多谐波频率。

5. I/O和外部存储器接口

Strtix Ⅲ FPGA的每个低电压CMOS和TTL标准可编程输入和输出引脚上有大量的标准接口。这些引脚包括复合IOE，它位于Stratix设备外围I/O模块中；包含双向I/O缓存区和I/O寄存器，允许引脚配置为全嵌入式双向单数据或双数据速率传输，其中用时钟的上升和下降沿加速数据传输。如图5-8所示，每行I/O模块有多达4个IOE而且每列I/O模块也有多达4个IOE；行IOE驱动行、列或直接链路互连，列IOE驱动列互连。许多标准的产品特性都支持，包括可编程的输入和输出延时、转换速率、总线状态保持和上拉电阻、漏极开路输出及大量片上串联终端模式。行I/O库支持I/O配置为132全双工1.25Gbps完全低电压差分信号通道（132Tx+132Rx）。

I/O结构也支持高性能外部存储器标准；支持高达400MHz频率的DDR存储器标准，如DDR3、DDR2和DDR SDRAM、QDR Ⅱ+和QDR Ⅱ SRAM和

RLDRAMⅡ。表 5-5 列出了典型存储器接口的速率等级。

表 5-5 Altera Stratix 的典型接口速度

内存接口标准 DDR SDRAM	I/O 标准	最大时钟速率	
		-4 速率等级	-2 速率等级
DDR2 SDRAM	SSTL-2	333	400
DDR3 SDRAM	SSTL-1.8	267	400
DDR3 SDRAM	SSTL-1.5	200	200
RLDRAM II	1.8V HSTL	250	350
QDRII SRAM	1.8V HSTL	250	350
QDRII+SRAM	1.5V HSTL	250	400

6. 千兆比特收发器

Stratix® Ⅲ 系列设备有一系列高速率的千兆比特收发器模块，它们使 DSP 系统中的数据可以在不同系统设备之间高速传输，即芯片到芯片传输。该收发器使用一对差分信号，即携带相反逻辑值的一对信号来传输数据，并且使用另一对来接收数据，因此取名收发器。这些收发器运行数据速率很高，比如 Stratix® Ⅲ 系列速率高达 1.25GHz 并且支持许多通信协议，如 Utopia 和 Rapid I/O™。这些会在后面解释。

传统系统使用总线分层连接系统设备来构建，因此设备在层级中根据它们要求的性能等级以适度的等级安放，即低性能设备安放在低性能总线等。许多实现个体连接性能要求的专门技术已经介绍过，比如提高总线频率或字长、分开处理事件及允许无序实现。这需要个体系统接口和复杂设计过程的进一步研发。在过去的几年中，随着允许低/高带宽的全频共享多点总线概念的演进，高速通信得到了长足发展。Rapid I/O™ 标准促进了这一个平台的发展，并且成为了高速率千兆比特收发器顶级成员。在运行中，主控或启动处理器生成一个请求事件，这个事件通过高速通信架构被传输到目标。然后这个目标生成一个回应事件返回到启动处理器来完成运行。Rapid I/O™ 事件被密封在数据包内，其中包括所有确定可靠传输到目标终点的必要的位域。Rapid I/O™ 提供同样的编程模块、处理机制和为了串行与并行实现的事件，包括基本映射 I/O 事件存储器、基于端口的消息传送和基于硬件连接的全局共享的分布式存储器（Bouvier 2007）。它也能处理任何结果的错误，应为每个数据包包括一个端对端循环冗余检测（Cyclic Redundancy Check，CRC）。因此 Utopia 和 Rapid I/O™ 通过为通信提供一个标准的接口减少了设计的复杂性。

7. 设备安全性

安全性是 FPGA 技术中一个主要的关注点，因为大多数主流 FPGA 设备都基

于SRAM技术，像标准存储器，内容是易读的。通常，设计者会使用供应商的专有软件创造出一个设计，生成一个配置数据文件，编写基于SRAM设备的程序。用EPROM或更一般的设备存储FPGA编程信息，它被储存在一个微处理器的系统存储器中而且在上电时加载。因此，配置信息数据能从EPROM或FPGA中的SRAM配置数据单元中读取，这种方式存在的问题是设计者的知识产权不能受到保护。

因此Stratix Ⅲ设备增加了一个特性，它通过使用工业标准AES算法将FPGA配置位流加密。AES算法通过一个储存在Stratix Ⅲ设备的安全秘钥实现，它被用于加密配置文件。当与一个外部主机，如微处理器，一起使用快速的、被动的、并行配置模式或当使用快速、主动串行或被动串行配置方案（Altera Corp. 2007）配置Stratix Ⅲ设备时，可以用来设计安全属性。设计安全属性可用于快速主动串行配置模式的远程更新。

5.3.3 Hardcopy®结构化ASIC系列

这些设备通过输出引脚、密集化、结构化来补足Stratix® Ⅱ设备，实现结构化ASIC。HardCopy®设备的重点是去掉可重新编程的FPGA逻辑、路由选择、存储器和与FPGA配置相关的逻辑，并通过金属直接连接取代SRAM配置资源。因此，设计者会使用Stratix® Ⅱ FPGA系列设计原型机，然后使用HardCopy®实现量化生产。

在这两种设备中，存储器、时钟网络和PLL是一样的，因为这些都是标准组件，但是Stratix® Ⅱ ALM和专用的DSP模块用逻辑模块的组合取代，即HCell。Quartus Ⅱ软件过去常用于实现设计，然后在设计被传给FPAG之前使用预制的HCell宏模块库来取代Stratix Ⅱ ALM与DSP配置。这是为了使用8个HCell宏模块实现8个可支持的运行模式，从而实现Stratix Ⅱ DSP模块的各种9、18、36位乘法、复合累加及复杂的乘法和加法。

HardCopy Ⅱ存储模块也能实现各种类型的存储器，可以具有或没有奇偶校验，包括真双端口、简单双端口和单端口RAM、ROM和FIFO存储器。HardCopy Ⅱ设备支持与Stratix FPGA同样的存储器功能和特性，如特殊的4k M4K RAM模块和512位的M-RAM模块。

当然，相比于传统的基于SRAM的FPGA实现，结构化的ASIC更具有吸引力的地方是它们不需要编程，因为可编程性已经被彻底移除了。因此，需要去掉一些特性，比如需要去掉加密数据流和配置状态引脚安全性等特性。除了有50ms的较短时延外，HardCopy Ⅱ结构化的ASIC遵循与ASIC上电一样的原则。在此期间，所有的寄存器都被复位；这是有吸引力的，因为成本的原因，在ASIC实现中不是所有的寄存器都可以复位。

5.4 Xilinx FPGA 技术

第一个 FPGA 是 1982 年开发的 Xilinx XC2000 系列。基本的概念是为了拥有可编程单元，并通过可编程构造连接，通过可编程 I/O 馈送信号，如图 5-13 所示。这个 Xilinx FPGA 不同于基于 PLD 的早期 Altera 设备，因此，Altera FPGA 没有同样的高级可编程的互连。这个架构包含的单元叫做逻辑单元或 LC，它的功能与 Altera LE 非常相似，如图 5-3 所示。这个互连是可编程的，并且基于 6 晶体管 SRAM 单元，如图 5-14 所示。通过定位单元在相互联系中的位置，能够通过水平面到水平面、垂直面到垂直面、垂直面到水平面和水平面到垂直面的连接实现灵活的路由。I/O 单元有大量的配置，允许大量接口模式，可将引脚配置为输入、输出或双向。

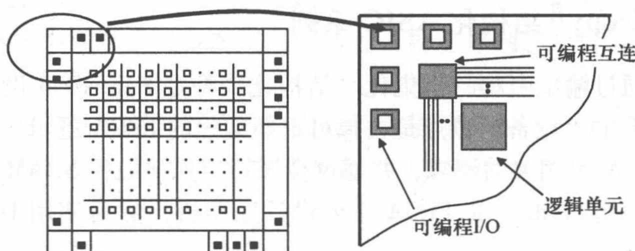


图 5-13 早期 Xilinx FPGA 架构

目前，根据摩尔定律，FPGA 成为在逻辑密度和速度方面不断扩展的胶合逻辑设备。该设备架构从 XC2000 一直升级到 XC4000，许多都未改变，比如，使用了同样的 LUT 表格尺寸。主要的演变是增加了快速加法器，制造商观察到通过在 LE 单元中添加一个额外的数据复用器，通过映射一些逻辑能进入快速进位加法器逻辑，一些能进入 LUT 实现快速加法器。图 5-15 所示为 Virtex™ FPGA 设备的原理。眼下，更大的系统设备仍然被认为是胶合逻辑，但是快速加法器逻辑的出现打开了有限范围 DSP 系统实现的可能性，特别是对乘法性能有要求的地方，但是不要求全系列乘数。这为大量早期基于 FPGA 的 DSP 实现技术打下了基础，将会在第 6 章介绍。

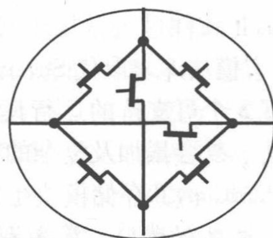


图 5-14 Xilinx FPGA SRAM 互连

此时，大量的 FPGA 产品制造商逐渐消失，然后开始了积累期（见表 1-1），在那个时期，FPGA 开始积累更复杂的组件，是始于板上专用乘法器，出现在第

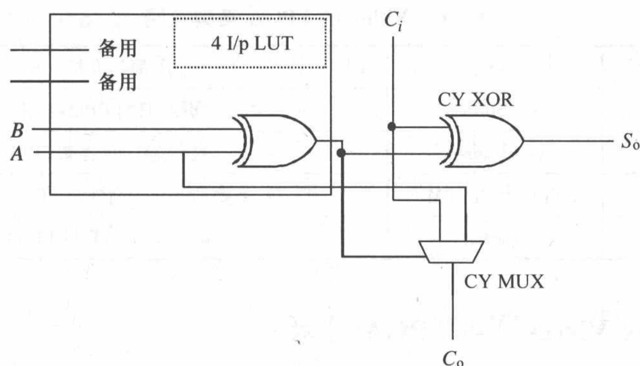


图 5-15　在 Xilinx Virtex™ FPGA 片中实现加法器

一个 Xilinx Virtex™ FPGA 系列里如图 5-16 所示；Power-PC 模块与千兆位收发器模块，出现在 Xilinx Virtex™-Ⅱ Pro；以及以太网 MAC，出现在 Virtex™-4。从图 5-16 中可看出，与 Altera 技术类似，Xilinx FPGA 越来越像一个以可编程为主要目的的 SoC，允许复杂处理模块连接在一起，用 LC 实现基本的逻辑功能。这个构造现在包含了标准系列的 LC，像以前一样允许把功能连接起来，但现在复杂处理模块，如 18 位乘法器和 PowerPC 处理器（见图 5-17），都越来越常见。平台 FPGA 概念现在被用来描述最近的 FPGA 设备来反映这个趋势。所有目前的 FPGA 系列都可从 Xilinx 购买，见表 5-6。

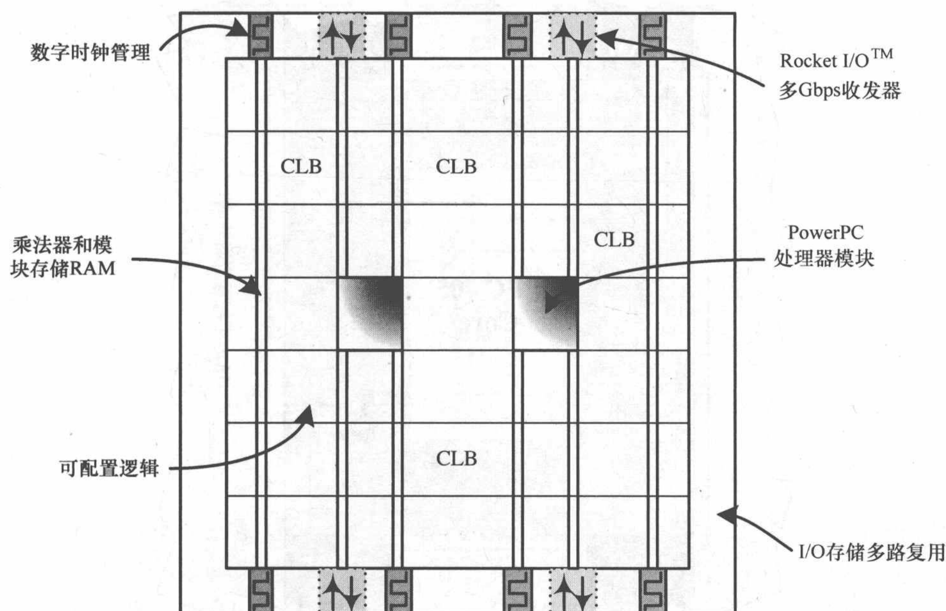


图 5-16　Virtex™-Ⅱ Pro FPGA 架构综述

表 5-6 Xilinx 的 FPGA 系列范围

类型	系列	简洁介绍
CPLD	XC9500XL	更老旧的 CPLD 技术
CPLD	CoolRunner	高性能，低功率 CPLD
FPGA	Virtex™/E/EM	主要的 Xilinx 高性能 FPGA 技术
FPGA	Spartan	低成本，高容量 FPGA

5.4.1 Xilinx Virtex™-5 FPGA 技术

本节集中于最新的 Virtex™ FPGA，即 Virtex™-5 系列，因为它是 FPGA 系列中演进的典型代表。没有介绍 CPLD 系列的原因是其细节内容可以在 Xilinx 网页上看到。Virtex™-5 有很多特色，如高性能逻辑的 LX 优化，具有低功耗串行连接的高性能逻辑的 LXT 优化，以及具有低功耗系列连接的 DSP 和内存密集型应用的 SXT 优化。XilinxVirtex™-5 系列有两个速度级性能增益而且时钟频率能达到 550MHz。它有大量的板上 IP 模块和大量的使 352 GMACS 性能最大化的 DSP48E 切片。它的引脚数也提高到了 600，具有 1.25Gbps LVDS I/O，以及在 100M~3.2Gbps 可选速率的串行通信连接 Rocket IO GTP 收发器。也包括了硬的 PCI Express 端点模块和 Tri 模式的以太网 MAC。

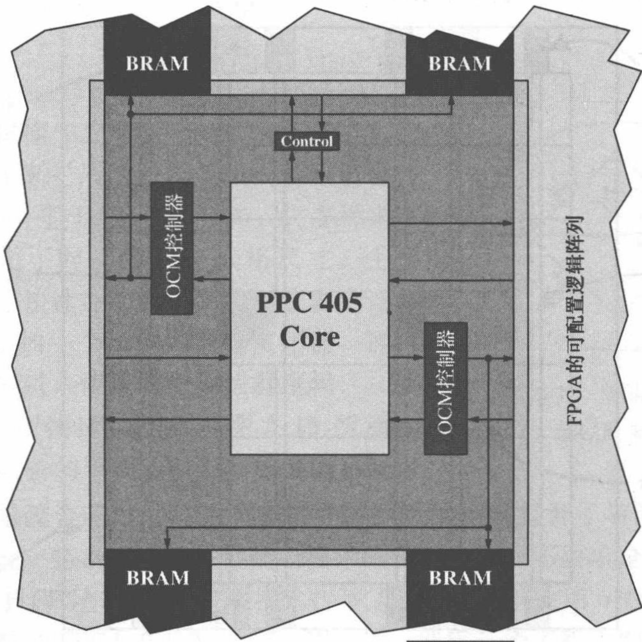


图 5-17 PowerPC 处理器模块架构

1. Virtex™-5 可配置的逻辑模块

在 Xilinx 设备中逻辑的实现使用了可配置逻辑模块（Configurable Logic Block，CLB）。为了实现一般路由选择矩阵的通道，每个 CLB 被连接到一个转换矩阵，并且包含一对并列的切片，每个都有独立的进位链，如图 5-18 所示。对每个 CLB 来说，在 CLB 底层的切片被标上 SLICE（0），在 CLB 顶层的切片被标上 SLICE（1）等。每个切片包含 4 个逻辑功能发生器（或 LUT）、4 个存储元件、多功能数据选择器及进位逻辑，而且因此能被认为包含了 4 个逻辑单元逻辑，如图5-19所示。除了这个，一些叫 SLICEM 的切片支持另外两个功能，即用分布式 RAM 存储数据及用 32 位寄存器移动数据。

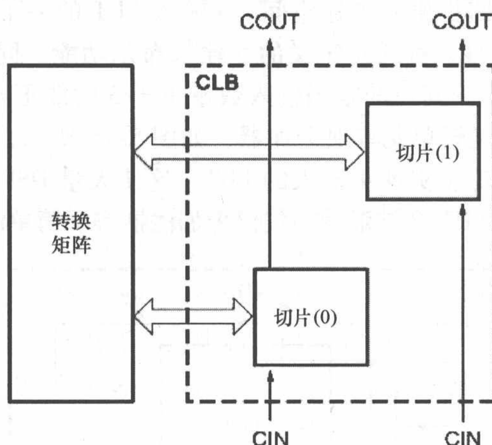


图 5-18　CLB 中切片的布置

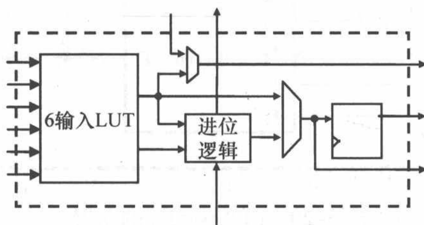


图 5-19　一个切片中的逻辑单元功能

基本逻辑单元的配置与 Altera LE 类似，如图 5-3 所示。它具有一个逻辑资源，一个连接到单触发器的 6 输入 LUT，中间经过多个数据复用器，以及执行快速加法的电路。与 Altera LE 元件一样，基本逻辑单元设计用来完成组合和时序逻辑实现，其中还有使用一个加法器的简单 DSP 电路。LUT 加上寄存器的基本

组合已被粘附在 Xilinx 架构中，并且已经从 Xilinx XC4000 系列和 Virtex™ -5 系列 FPGA 的 4 输入 LUT 扩展到了 6 输入 LUT。这是摩尔定律支配下科技提升的一个反映。现在 Xilinx Inc.（2007a）争论的 6 输入而非 Rose 等人（1990）研究的 4 输入 LUT，体现出设计中提高硅面积利用率的关键路径。LUT、触发器（Flip Flop FF）和一些特殊器件，如进位链、专用复用器及连接这些器件的方式，被称为 ExpressFabric 技术。

CLB 能够实现以下功能：纯逻辑功能使用 6 输入 LUT 逻辑及数据复用器绕开寄存器；简单寄存器使用数据复用器直接将数据馈入和馈出；以及时序逻辑电路使用 LUT 馈入寄存器。使用数据复用器来创建更大的 LUT 和寄存器，为更大的组合和时序逻辑电路提供了创建空间。6 输入 LUT 的一个特征是它有两个输出，这就允许 LUT 实现两种任意定义的 5 输入布尔功能，同时这两个功能共享输入，如图 5-20 所示。其目的是当输入数量小于 6 时能更好地利用 LUT 资源。这个概念也允许使用逻辑单元实现全加器，如图 5-15 所示，同时，使用额外输入和输出来为其他的功能实现 4 输入的 LUT。这使大量 DSP 应用中的硬件可以更好地被利用，否则 LUT 会被浪费在仅仅为加法器实现简单门功能上。

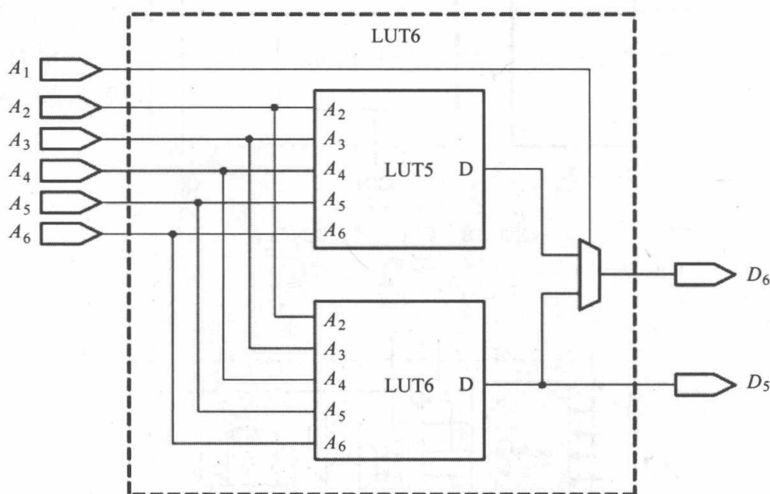


图 5-20 CLB 中切片的布置

与 Altera 技术一样，寄存器资源也很灵活，允许存储的潜能很大，从 D 型边沿触发器到电平触发的锁存器，都有各种各样的同步异步输入的时钟、时钟使能、置位/复位。D 型触发器的输入能直接由许多源驱动，包括 LUT 输出、其他的 D 型触发器和外部输入。

在 XilinxVirtex™-5 设备中，LUT 的更大优势是它可以提供更大的分布式

RAM 模块和 SRL 链。不同分布式存储器配置的样本见表 5-7，它列出了用于创建不同存储器的配置数量。分布式 RAM 模块具有同步写资源，而且使用同一切片的触发器，能实现同步读取。通过减少时钟到输出的时延，改进了关键路径，但是增加了一个额外的时钟循环延迟。

表 5-7　各种存储配置的 LUT 数量

存储单元数量	存储类型		
	32	64	128
单端口	1（1-位）	1（1-位）	2（1-位）
双端口	2（1-位）	2（1-位）	4（1-位）
四端口	4（2-位）	4（2-位）	
简单双端口	4（6-位）	4（3-位）	

大量的存储配置已经列了出来。对于单端口配置，一个公共的地址端口可用于同步写和异步读。对于双端口配置，分布式 RAM 有一个同步写和异步读端口，它连接到一个函数发生器和一个异步读端口，以及第 2 个函数发生器。在简单双端口配置中，写端口没有读的功能。4 端口配置在异步读创建 3 端口和 3 个函数发生器概念的基础上进行扩展，加上一个同步写和异步读端口，一共有 4 个函数发生器。

下一节将讲述对更大存储模块的理解，但是 Altera FPGA 提出了以更小的分布式 RAM 与更大的 RAM 模块的组合来实现同样分级存储器体系的概念，并且在一定范围内被公认。LUT 能实现一个 ROM 功能，从而实现可编程移位寄存器，如第 6 章所述。Virtex™-5 的函数发生器和关联的一些如图 5-19 所示的数据复用器，能使用一个 LUT 实现一个 4∶1 的数据复用器，使用两个 LUT 实现一个 8∶1的数据复用器。

2. 存储器构成

除了分布式 RAM 外，Virtex™-5 设备有大量的 36KB RAM 模块，它们每个包含两个独立受控的 18KB RAM。整个存储器的配置见表 5-8。在不使用可编程互连的前提下，18KB RAM 模块可以被配置成 36KB RAM。模块 RAM 布置成纵列，而且能够串联来创建更深更宽的 RAM 模块。每个 18KB RAM 模块，双端口存储器包含一个 18KB 储存空间和两个完全独立的接入端口，连同其他电路一起，可以用来实现预期的 RAM 功能，如图 5-21 所示。存取引脚的完整定义在下面给出，这代表了标准 RAM 配置。

表5-8 Virtex™-5的类型和使用

内存类型	位/模块	模块编号	建议用法
分布式RAM/切片	1024	14720	移位寄存器、小FIFO缓冲区，滤波器的FIFO缓冲区，滤波器，延迟线
36kbit RAM模块	18000	488	多速率FIFO

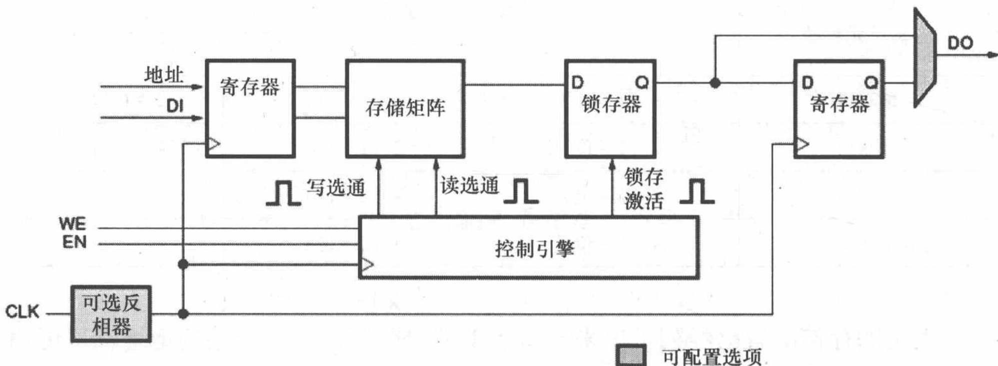


图5-21 模块RAM逻辑框图（Xilinx Inc. 2007b）（由Xilinx Inc. 许可复制）

1）每个18KB模块RAM的时钟能够被配置为上升沿或下降沿有效，所有输入和输出端口都参考这个时钟；

2）使能信号不能通过未激活的使能引脚控制端口的读、写和置位/复位功能，存储器将保持先前的状态；

3）附件使能信号叫作字节宽写使能信号，与使能信号共同控制RAM的读和写；

4）寄存器使能引脚控制可选输出寄存器；

5）置位/复位引脚可以强制数据输出锁存器置位；

6）为读或写选择存储单元的地址总线，它的数据位宽由选择的RAM功能的规模决定。

在锁存模式下，读地址寄存在读端口，并且存储的数据在RAM存储访问时间后被加载到输出锁存器里。当使用输出寄存器时，读操作将带来一个附加的延迟周期。写操作也是一个单时钟沿操作，将写地址寄存在写端口，并且输入数据被储存在存储器里。这个附加的电路如图5-21所示，指明反向时钟与一个寄存器的输出是如何同时被支持的。使用INIT参数能初始化RAM的内容，而且能通过HDL源代码体现出来。

RAM具有许多的配置选择，其中一些见表5-9，表中列出了数据位宽是如何根据存储器深度，即存储单元的数量来权衡的。

表 5-9　Xilinx Virtex™-5 模块的存储容量

数据宽	存储深度
1（级联）	32768（65 536）
2	16384
4	8192
9	4096
18	2048
36	1021
72	512

模块 RAM 使能中添加了专用逻辑来允许同步或异步先入先出的创建；这在一些高级设计方法里很重要，正如我们后面将看到的。专用逻辑可以替代更慢的可编程 CLB 逻辑和路由资源，而且为写和读指针的生成及与 FIFO 相关的各种标记的设置提供必要的硬件。FIFO 的容量是可以被推算出来的，包括 8KX4，4KX4，4KX9，2KX9，2KX18，1KX18，1KX36，512X36 和 512X72。

3. Virtex™-5 DSP 处理资源

除了 CLB 中可扩展的加法器，Virtex™-5 还有专用 DSP 处理模块，叫作 DSP48E。Virtex™-5 有可达 640 个的 DSP48E 切片，它们分布在 FPGA 的不同位置，而且支持大量的独立功能，包括复合 MAC、复合加法、3 输入加法、快速移位、宽总线复用、量级比较器、位宽逻辑功能、模式检测和宽计数器。这个架构也允许连接复用 DSP48E 切片来形成一个更宽的 DSP 功能范围，比如 DSP 滤波器、相关器和频域功能。

简易版本的 DSP48E 处理模块如图 5-22 所示。DSP48E 模块的基本架构是一个乘法 - 累加核，它对大量的 DSP 计算来说是一个非常有用的引擎。但是，除了基本 MAC 功能外，DSP48E 模块也允许大量其他的运算模式，如下：

1）25 位 × 18 位乘法运算，它可进行流水线应用；

2）96 位累加或加法或减法器（跨过了两个 DSP48E 切片）；

3）三倍的和有限的四倍加减法；

4）专用按位逻辑运算；

5）对上溢和下溢算法的支持。

每个 DSP48E 切片有一个 25 位 × 18 位的乘法器，它是从两个数据选择器馈入的，数据选择器接收一个 30 位的 A 输入和一个 18 位的 B 输入，它们要么来自交换矩阵，要么直接来自 DSP48E 的下方。在被馈入到乘法器之前，这些能被存储在寄存器里（图 5-22 中没有显示）。仅仅在乘法运算之前，A 信号被分开，而且仅有 25 位信号被馈入乘法器中。使用一个快速乘法器技术，它产生一个等价

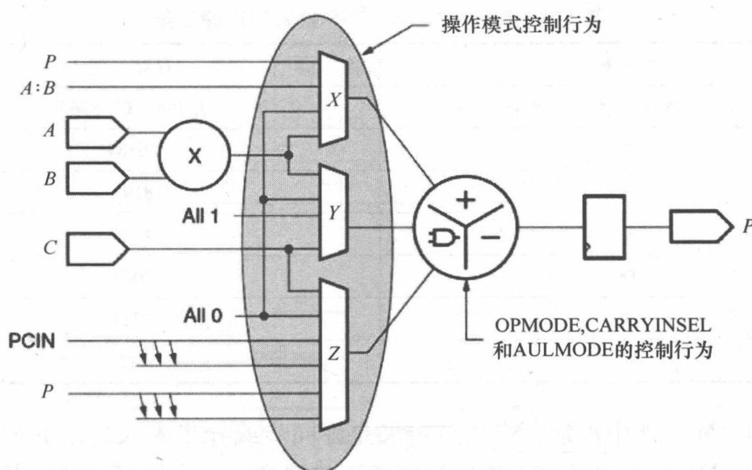


图 5-22 名为 DSP48E 的 DSP 处理模块（Xilinx Inc. 2007c）（由 Xilinx Inc. 许可复制）

的两部分乘积形式的 43 位二补数结果，然后分别扩展为有 48 位有符号数的乘数 X 和乘数 Y，最后，馈入到三输入加法/减法器中实现最终的求和。

如第 4 章所述，许多快速乘法器设计概念都相同，这个概念就是使用快速进位保留加法器来最终生产一个最后的总和与进位信号，并使用一个快速进位脉冲来实现最后的加法。最后的加法代价是很高的，要么是在速度方面的代价，而如果已经使用了速度提升的技术，那就是面积方面的代价。通过推迟加法到 ALU 阶段，这一个两阶段加法可以通过乘法积累实现，这需要通过执行一个三阶段加法来计算最后的乘法输出并对累加输入执行一阶段加法实现。再次，为了灵活，加法/减法器单元已经扩展增为 ALU，因此使用小的硬件开销就可以实现更多的功能。因为传统乘法的最后一个阶段是在第二阶段加法器中执行的，所以如果需要，则可以利用一个三输入加法与第三输入一起完成 MAC 运算。

数据复用器允许附加更多灵活性等级。比如，P 输入能被用于馈入一个输入，该输入可以来自 Z 数据复用器中使用 PCIN 的 DSP48E 模块，也可以由当前的 DSP48E 模块回送。比如说，在递归时使用 P 输入到 Z 数据复用器，如果不需要乘法器，则可以通过使用 A: B 输入绕过，它是两个输入信号 A 和 B 的串联，它们分别是 25 位和 18 位，这个字长为 43 位，同乘法器输出一样。初始化输入到 ALU 的值可以规定为全 0 或全 1。为了提高单元的灵活性，加法器也能被分成几个更小的加法器，即允许执行两个 24 位加法或 4 个 12 位加法。这就是所谓的 SIMD 模式，为了用多个数据做简单的加法运算，提出了 SIMD 操作。DSP48E 切片可以实现 17 位右线移位，允许一个 DSP48E 切片的部分乘积项被移动到右边并添加到下一个部分的乘积项上，在邻近的 DSP48E 切片里进行计算。当专用乘法器被用作构建模块时，这个功能可以构建更大的乘法器。

图5-22中仅画了基本功能，除了乘法或乘法累加输出 P 外，其他信号没有表示出来。包括：

1）叫作ACOUT的可级联的 A 数据端口，它允许 A 的内值被直接馈入到输出，假设 A 信号已经在内部延时了，这会为DSP功能提供延时链，比如FIR滤波器；

2）当跨过两个DSP48E切片支持96位加法/减法时，作为内部信号的可级联的进位输出（CARRYCASCOUT）和符号（MULTSIGNOUT）信号被用于表示进位输出和符号；

3）附加的SIMD模式需要多达4个进位输出信号（CARRYOUT）来支持，这里需要多达4个独立加法器来生成这些进位信号。

4）模式检测器支持许多数字收敛的舍入、上溢出/下溢出、模块浮点，还支持带模式检测器输出（PATTERNDETECT和PATTERNBDETECT）的累加器终端计数（计数器自动复位），来显示模式是否匹配，并分离上溢出（OVERFLOW）和下溢出（UNDERFLOW）信号。

从功能角度，综合工具很大程度上隐藏了设计功能是如何映射到FPGA硬件上的，但是理解可用功能等级是很重要的，因为它决定了用户将采取的设计方法。许多细节例子都列在了用户相关指南里面（Xilinx Inc. 2007c），详细说明了性能指标是如何达到的。

4. 时钟网络和PLL

XilinxVirtex™-5 FPGA系列能提供550MHz的时钟频率。Virtex™-5 FPGA的时钟包括6个时钟管理模块（Clock Management Tile，CMT），每一个CMT包含两个数字时钟管理器（Digital Clock Manager，DCM）和一个PLL。总之，FPGA合计有18个时钟发生器。

DCM是XilinxVirtex™-5 FPGA的一个关键特征，它具有范围广大的强大时钟管理特性，包括一个延时锁定回路（Delay Locked Loop，DLL），这可以使输入的时钟和早先介绍的已经产生的时钟对齐。它也可以产生大范围的时钟频率，包括一个两倍频和一系列特定的分数级时钟频率，可以是输入时钟的1.5，2，2.5，3，3.5，4，4.5，5，5.5，6，6.5，7，7.5，8，9，10，11，12，13，14，15或16倍。支持粗粒度（90°，180°与270°）与细粒度相移，以及各种精细的或分数级的相移。

PLL的主要作用是与DCM一起实现频率合成，以及内部或外部时钟消抖。对于时钟发生器，6个PLL输出计数器被复合成一个单一时钟信号来为DCM提供参考时钟信号。来自PLL的两个输出时钟能驱动DCM，例如当一个能驱动第一个DCM时另一个可以驱动第二个DCM。每个DCM的输出可以灵活地被复合到一个单一时钟信号中，作为PLL的一个参考时钟使用，而且DCM随时可以被

用作 PLL 的参考时钟（Xilinx Inc. 2007c）。

5. I/O 和内部存储器接口

与 Altera FPGA 设备相比，Virtex™-5 FPGA 支持许多不同 I/O 标准接口，称为 SelectIO™驱动器和接收器，允许控制输出强度和转换速率及片上端接。与 Altera FPGA 一样，I/O 接口被放置在一个包括 40 IOB 的库中，覆盖了一个 20CLB 的高物理区域，而且由单一时钟控制。Virtex™-5 FPGA 也有数控电阻（Digitally Controlled Impendance，DCI）技术，可以调节输出阻抗或输入传输线，准确匹配传输线的特征阻抗。有效端接 PCB 印制线信号在高速电路的实现中日趋重要，而且这个方法可以避免增加板上终端电阻。同时还支持许多标准，包括低电压晶体管对晶体管逻辑（Low Voltage Transistor-Transister Logic，LVTTL）、低电压互补金属氧化物半导体（Low Voltage Complementary Metal Oxide Semiconductor，LVCMOS）、外围组件接口（Peripheral Component Interface，PCI），包括 PCIX、PCI33、PCI66 和低电压差分信号等。

Virtex™-5 FPGA 还支持输入串行并行转换器（Input Serial to Parallel Converter，ISERDES）和输出并行串行转化器（Output Parallel to Serial Converter，OSERDES）。这些允许非常快的外部 I/O 数据速率，比如 SDR 和 DDR，来馈入到要慢上一个数量级的内部 FPGA 逻辑。这本质上是一个附加的串行并行的转换硬件模块，它允许进入到 FPGA 架构的并行数据流序列重排，以及处理选通脉冲到 FPGA 时钟域交叉电路的存在。

5.5 Lattice FPGA 系列

Lattice®提供许多 FPGA 架构，包括基于 CPLD 概念的 ispXPLD™ 5000MX 系列，具有高速通信接口、存储器和专用 ASIC 模块实现的更加标准化的 FPGA 的 LatticeSC/M 系列，具有许多与 LatticeSC/M 系列相同功能的低成本 LatticeECP2/M 系列。

5.5.1 Lattice® isp XPLD 5000MX 系列

isp XPLTD™5000MX 系列通过 E^2PROM 非易失性单元存储设备配置，SRAM 提供逻辑实现，提供启动时逻辑可用性解决方案。它还具有灵活的存储能力、支持单端或双端 SRAM、FIFO、三元的内容可寻址存储器（Content Addressable Memory，CAM）的运算及专用运算功能。但是，该技术主要受益于使用 CPLD 架构实现可预测定时。ispXPLD 5000MX 设备的架构由与一个全局布线区相连的多功能模块（Muti Function Block，MFB）单元组成，这些多功能模块经多个共享数组（Multi Sharing Array，MSA）连接到输入和输出引脚，如图 5-23 所示。

MFB由多功能数组和相关的路由组成，它能够处理来自GRP和4个全局时钟及复位信号的多达68个输入并且产生到宏单元或别处的输出。该设备允许相邻MFB级联来支持更宽的运算，每个MFB能配置成许多模式，包括逻辑和存储器配置，比如单端和双端RAM、FIFO模式及CAM。

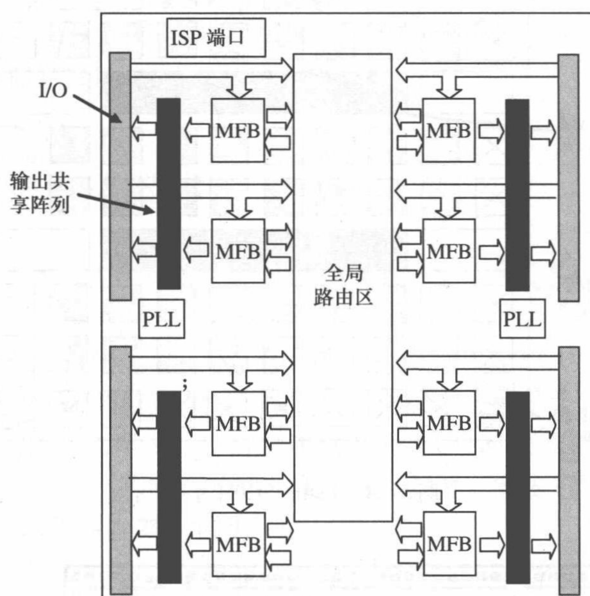


图5-23　Lattice isp XPLD 5000MX系列

这里集中介绍LatticeSC/M FPGA（Lattice Semi. Inc. 2007），因为它代表了高性能的FPGA系列，具有许多出现在更低成本Lattice ECP2/M系列上的特点。LatticeSC/M设备的架构如图5-24所示，它把包含的标准可编程I/O模块连接到几排逻辑模块，形成可编程功能单元（Programmable Function Unit，PFU）。这些PFU由切片组成，每个都包含两个4输入LUT和两个寄存器，与进位传播和进位生成信号一起形成快速加法器结构。与其他FPGA产品一样，这个功能能够实现组合和时序逻辑，按要求缩放LUT大小和寄存器容量。与ispXPLTD™ 5000MX系列类似，PFU也能配置为各种存储类型。最大的设备具有115k的LUT。

除了可编程逻辑外，该设备还包含可达7.8Mbit的嵌入式模块RAM，来匹配PFU中的2Mbit分布式RAM。这些系统成员如同其名称一样，可以配置成RAM、ROM或FIFO，具有高度可编程性。除了标准可编程I/O引脚外，高性能I/O作为专用SERDES和PCS硬件也被包括在内，具有2Gbit/s I/O能力。这实现了在IP核上聚集较大的专用IP功能的概念。

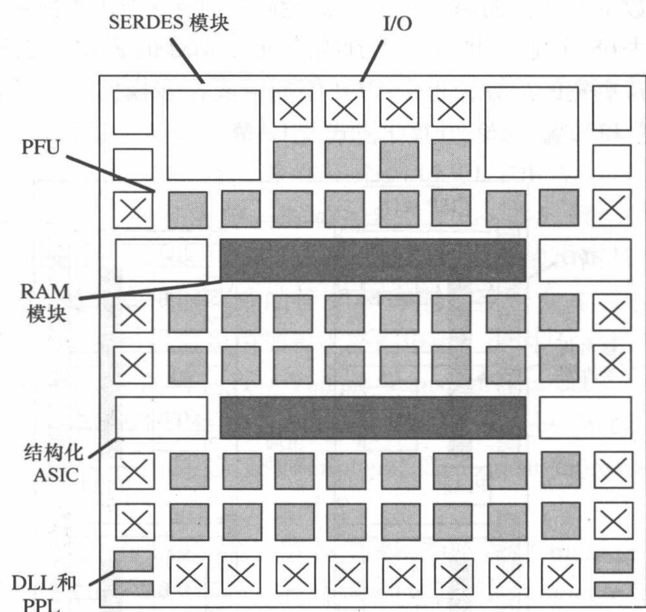


图 5-24 LatticeSC/M 系列

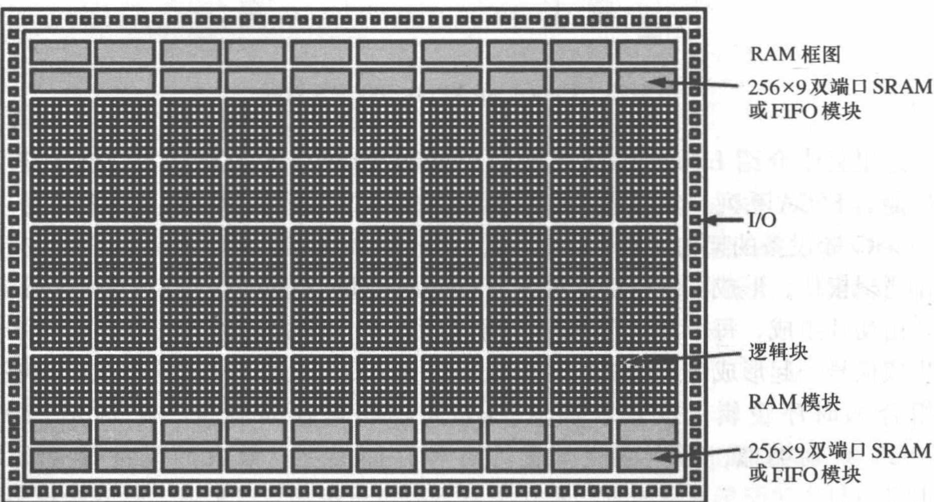


图 5-25 ActelPro ASICPLUS FPGA（Actel Corp. 2007b）（由 Actel Corp. 许可复制）

与 Altera 和 Xilinx 的产品不同的是结构化的 ASIC 性能，如图 5-25 所示。这个成本屏蔽矩阵（Masked Array for Cost Optimization，MACO）模块，具有 50000

个 ASIC 门，使用 90nm CMOS 处理技术实现，并做了速度、功率损耗和面积的优化（Lattice Semi. Inc. 2006）。MACO 模块直接接到 I/O 和 FPGA 结构上，因此针对特殊硬件需求可以完成专用的快速低功耗实现。除了门外，每个 MACO 模都包含 3 个 64 × 40 的异步双端口 RAM，以及 FPGA 的 RAM。双端口 RAM 与 MACO 模块处于同一位置而且能通过专用 MACO 接口模块或 MIB 进行存取。通过布置于 I/O 引脚和板上模块 RAM 之间，可以实现大量的快速、低功耗模块，如专用或特殊存储器接口。

5.6 Actel® FPGA 技术

Actel® 提供大量 FPGA 技术，都是基于闪存和反熔丝技术。他们最近正式推出了一个名为 Fusion™ 的技术，它代表了第一个混合 FPGA 信号（Actel Corp. 2007a）。它包括 A-D 转换器、嵌入式闪存器，以及以 D 型触发器和 RAM 的形式出现的更加传统的数字 FPGA 硬件。闪存技术是非易失性的，意味着 FPGA 会存储其设计且上电即可，不需要通过 ROM 或协同处理器编程。最大的设备拥有 1.5M 个系统门，具有 270kbit/s 的双端 SRAM、多达 8Mbit 的闪存、1kbit 用户闪存 ROM，以及多达 278 个用户 I/O。

5.6.1 Actel® Pro ASIC^{PLUS} FPGA 技术

Pro ASIC^{PLUS} FPGA 技术是基于闪存技术的。它与 E^2PROM 技术相似，通过储存电荷实现程序的存储。FPGA 技术通过与门阵列类似的一种架构组件，并且包含大量的单元，且每个单元能被配置成一个 3 输入逻辑功能，或者一个 D 型触发器，如图 5-25 所示。架构包括网格的单元，大量嵌入式双端口 SRAM 模块，以及允许同步和异步运算的顶层和底层。这些单元由大量数据复用器和逻辑门组成，这足够制作一个连接必要全局复位和时钟信号的触发器。这些单元同时连接到本地和更远的线路路由。

这些单元通过分层路由进行连接，并且按照长度分为 4 个等级，这就像一个细颗粒架构，与老的 Xilinx XC2000 FPGA 技术相似。下一级的线路管理 1 个、2 个或 4 个单元，它们可以通过单元的输出端访问。再下一级互连是跨越了芯片长度的长线资源。最后一级是全局信号，它们必须定义为低延时，因为它们是全局信号，比如时钟、复位和使能信号，它们在全局使用容易降低芯片的性能。

该设备还包含两个时钟调节模块，和其他 FPGA 设备一样，包括 PLL、用于同步外部信号的延时电路、乘法器/除法器电路，以及连接全局路由网络的必要电路。该设备具有 4 个时钟网络或全局树，特别设计用于分配低延时、低抖动的

时钟信号。

实现如此细粒度技术的关键是硬件中各设计单元很好的布置，因为有限的路由分配会妨碍设计的质量和底层硬件模块的利用。因为这个原因，软件就允许使用限制来控制设计布置。通过生成一个电路架构来匹配最佳的 FPGA 资源要求的概念显然是必要的，不仅有益于获得必要硬件性能，而且有利于减轻有效实现里的软件工作。与其他 FPGA 一样，有一系列的 I/O 模块可供使用，而且具有针对板级系统内容的 JTAG 形式边界扫描。

5.6.2 Actel®反熔丝 SX FPGA 技术

除闪存设备外，Actel 也有反熔丝技术，它是一次性编程的。反熔丝是一个双端点器件，在未编程状态呈现出高阻抗。如图 5-26 所示，两个导体一般通过绝缘体分开，有时通过无定形的或“可编程的”硅分开。当一个高电压加到终端之间时，反熔丝会熔断并且形成一个低阻抗连接（与熔丝的熔断电路相反）。

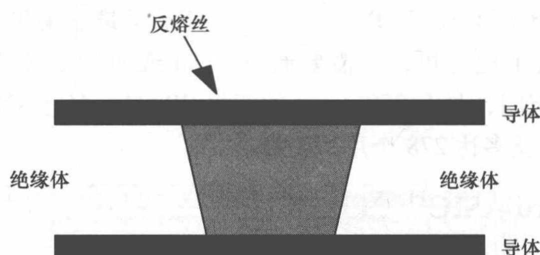


图 5-26 基础反熔丝连接

如图 5-27 所示，在 Actel 反熔丝连接的情况下，连接通过非晶硅和介电材料的组合实现，并且导通状态下有 25Ω 阻抗和 1.0fF 的电容（Actel Corp. 2007c）。如图 5-27 所示，采用三层金属与金属对金属的反熔丝，使性能更好且空间更小。金属对金属的反熔丝降低了编程阻抗，因此速度更高，并且多金属层现在允许反熔丝布置在逻辑之上，因此省去了路由通道。这使逻辑密度可以更高，使高性能小封装得以实现。

该架构包括了许多像 Pro ASIC^{PLUS} FPGA 技术的模式，这种技术粒度很细，且有着两种类型的单元，即 C 单元和 R 单元，C 单元实际上是一个组合单元，它包括一系列数据复合器和逻辑门，而 R 单元是一个寄存器单元，它包含一个有着各种乘法硬件的 D 型触发器，这就允许各种输入连接，而且允许使用各种时钟信号为单元提供时钟。由于配置单元的可编程性，结构中包含各种不同的单元，因此必须创建单元的预制映射，称为簇。图 5-28 所示为两种类型的超级簇，

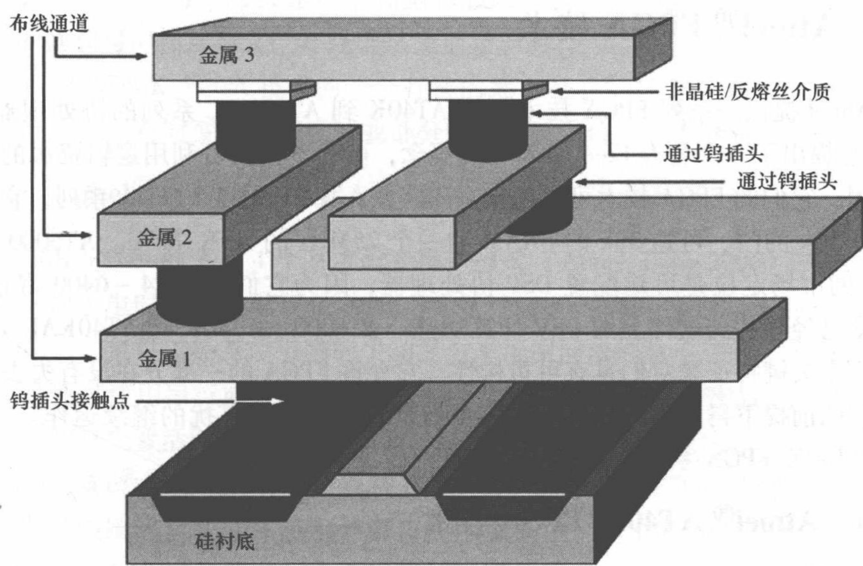


图5-27 Actel® Pro ASIC^{PLUS} FPGA（Actel Corp. 2007c）（由 Actel Corp. 许可复制）

一个超级簇包含了两个CRC簇，另一个超级簇包含了一个CRR簇与一个跟随其后的CRC簇。

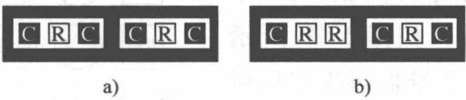


图5-28 簇结构
a）类型1超级簇 b）类型2超级簇

在该结构中，也给出了大量不同级别的路由。第一种路由包含一种直接连接，用于横向路由，实现从一个簇中的C单元到与它相邻的R单元的连接；还包含一种快速连接，用于纵向路由。同时具有两种全局导向路由资源，即分段路由和高驱动路由，就像它们的名字一样，是为了更小的路径，例如分段或更全局的路径。与Pro ASIC^{PLUS} FPGA的技术一样，为了执行有效，必须考虑单元的详细布置，否则就会受到可编程路由的时延缺陷影响。

300MHz的时钟速率引用自Actel Corp.（2007c）。此外，它还论述了安全是技术的一个关键特色，因为它证明了逆向工程很难做到，其原因是很难分辨出已编程和未编程的反熔丝，并且没有可拦截的配置位流。

5.7 Atmel® FPGA 技术

Atmel 提供一系列 FPGA 技术，从 AT40K 到 AT40KAL 系列的协处理器 FPGA，它提出了一个名为 Free RAM™的概念，可在不侵占可利用逻辑资源的情况下使用。它们的 FPGA 技术可被用作嵌入式核，如 FPSLIC™ FPGA 系列，它提供了 5k~50k 的门、可达 36k 的 SRAM 和一个 25MHz 的 AVR MCU。AT6000 系列 FPGA 的市场定位是可重配置 DSP 协处理器，因为它们有 1024~6400 寄存器，很适合已经移植到硬件上的 DSP 计算功能。AT6000、AT40K 和 AT40KAL FPGA 系列一个关键特征是它们具有可重构性，它允许 FPGA 的一部分在没有失去寄存器数据的前提下得到重构，同时 FPGA 剩余的部分可无打扰的继续运作。因此，将对 AT40K FPGA 系列做较详细的介绍。

5.7.1 Atmel® AT40K FPGA 技术

AT40KAL 是一个基于 SRAM 的 FPGA，具有分布式双端/单端 SRAM 和 8 个全局时钟，以及一个全局复位。这个系列有 5000 ~ 50000 个可用的门。AT40KAL 是一个细粒度 FPGA 架构，包含编织成 4×4 格状的简单单元，每一个都被中继器单元包围，如图 5-29 所示 (Atmel Corp. 2006)。中继器再生了信号并允许在同一平面内任意总线到其他总线的连接。每个中继器连接两个相邻的本地总线分段，实现了图 5-29 中 4 个单元的局部连接，并拥有两个快速总线分段，实现横跨 8 个单元的更长线路连接。

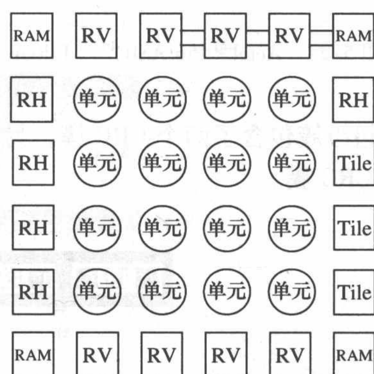


图 5-29 Atmel AT40K FPGA 4×4 单元

核心单元的粒度非常细，它包括两个 3 输入 LUT（8×1 ROM），也可以被配置成一个 4 输入 LUT、一个 D 型触发器、一个 2 选 1 的数据复用器和一个与门来实现乘法器阵列。和其他 FPGA 技术一样，LUT 和 D 型触发器的结合允许各种组合和时序逻辑实现。有一种 DSP 模式，但是相较于其他技术而言它是较基础的，因为它仅允许使用那个单元生成一个阵列乘法器，因此相对于其他 FPGA 技术，其 DSP 性能较差。

在每个中继器行列的交叉点上，有一个通过相邻的总线访问的 32×4 RAM

模块，通过一系列局部的快速水平和垂直总线，它能被独立寻址。双端口 FreeRAM 的读和写都相互独立，并且读是完全异步的。

Atmel® AT40K FPGA 技术的一个有趣的地方是它能被部分重构，也就是被编程，允许在运行时变更全部设计或部分设计。这点值得详细分析。

5.7.2 Atmel® AT40K FPGA 的重构技术

AT40K FPGA 技术有 4 个基本配置操作模式（Atmel Corp. 2006）。第一，当设备第一次上电时，有上电复位功能，实现对所有内部配置 SRAM 的一个完全复位；第二，也可以经复位引脚的手动复位来调用同样的复位序列；第三，配置下载，是通过使用串行或并行数据经过输入引脚来对 FPGA 的配置 SRAM 编程；第四，当没有配置活跃时，AT40K FPGA 允许字级别的完全重构。

CacheLogic®架构是允许用户重构 FPGA 的一部分，而 FPGA 的剩下部分可以不受影响的继续运行，这是使用开窗口技术做到的。它允许用户把 SRAM 存储器映射加载到更小的分段中，允许用新的配置信息重写配置 SRAM 中没有被使用的部分。在同步 RAM 模式中，设备接收一个 32 位或 40 位宽的位流，包含一个 24 位地址和一个 8 位宽或 16 位宽的数据。地址、数据和写使能在 CCLK 的上升沿同时有效。在这个模式下，设计与一个微处理器的通用 I/O 口对接，FPGA 配置 SRAM 可以看成是一个简单的存储映射地址空间。用户拥有到整个 FPGA 配置 SRAM 的全部读写通道。消除了通常与位流联系的开销，会实现更快的重配置。

5.8 FPGA 技术上的总思考

本章涵盖了来自各公司大量的 FPGA 技术，重点突出了两个主要的供应商，也就是 Xilinx 和 Altera。FPGA 技术的主要驱动力已经从一个使用 LUT 和可编程互连实现简单逻辑的细粒度设备，转移到了异构复杂单元的集合，比如专用 DSP 模块、高速通信模块、软和硬处理器，以及前面提到过的 LUT。

这已经对设计进程有了许多的影响。与以前的 FPGA 相比，目标已经成为了开发一个有效的实现，FPGA 硬件的利用已经成为了主要关注点。这涉及使用最初的原理图设计捕获包，以及最近的基于 HDL 的工具，来达到一个有效的设计实现。考虑到主要目标是取得高利用率，底层 LUT 的利用耗费了相当大的努力。因此，开发了许多电路级设计技术来最优化地使用 LUT 和触发器，以此来完成实现。这些技术在第 6 章都会更加详细地介绍，因为它对原理理解很重要，尽管

综合工具中已有相当多的介绍。因此，对设计技术做了简述。

复杂性已经随着较新的FPGA增长起来，主要的挑战已经是在一个层面上设计电路架构（见第8章），在另一个层面上设计系统级架构，这是第7、9和11章中的重点。一般电路架构的重点包括源于DSP算法的SFG表示的所有FPGA的具体细节。因为在DSP实现领域里FPGA的卓越表现，所以这里利用并行和流水线分析了并发性。这个过程的实现在第8章中会有详细的介绍，高级工具流在后面的章节也会有介绍。

参考文献

Actel Corp. (2007a) Fusion family of mixed-signal flash FPGAs with optional soft arm support. Web publication downloadable from www.actel.com.

Actel Corp. (2007b) Proasicplus flash family FPGAs. Web publication downloadable from www.actel.com.

Actel Corp. (2007c) Sx family FPGAs. Web publication downloadable from www.actel.com.

Altera Corp. (2000) Max ii device family data sheet. Web publication downloadable from http://www.altera.com.

Altera Corp. (2007) Stratix iii device handbook. Web publication downloadable from http://www.altera.com.

Atmel Corp. (2006) 5k–50k gates coprocessor FPGA with freeram. Web publication downloadable from http://www.atmel.com.

Bouvier D (2007) Rapidio: the interconnect architecture for high performance embedded systems. Web publication downloadable from http://www.techonline.com/.

Lattice Semi. Inc. (2006) Delivering FPGA-based pre-engineered IP using structured ASIC technology. Web publication downloadable from www.latticesemi.com.

Lattice Semi. Inc. (2007) Latticesc/m family data sheet. Web publication downloadable from www.latticesemi.com.

Rose J, Francis RJ, Lewis D and Chow P (1990) Architecture of field programmable gate arrays: the effect of logic block functionality on area efficiency. *IEEE Journal of Solid State Circuits* 25(5), 1217–1225.

Xilinx Inc. (2007a) Achieving higher system performance with the virtex-5 family of FPGAs. Web publication downloadable from www.xilinx.com/bvdocs/whitepapers/wp245.pdf.

Xilinx Inc. (2007b) Virtex-5 user guide. Web publication downloadable from www.xilinx.com.

Xilinx Inc. (2007c) Xilinx ug193 virtex-5 xtremedsp design considerations: User guide. Web publication downloadable from www.xilinx.com/bvdocs/userguides/ug193.pdf.

第 6 章　FPGA 实现详解

6.1　引言

先前的章节已经在背景方面对 DSP 和计算机算法作了介绍，然后在最后两章，将突出讲述各种实现技术；第 4 章已经强调了更加广泛的技术并且第 5 章已经稍加详细地描述了各种 FPGA 产品。剩余的章节将描述将复杂的 DSP 系统实现到多样化平台甚至一个单个 FPGA 设备上的问题。其中包含一些考虑事项，比如 DSP 系统实现的合适模型的选择、硬件和软件中 DSP 复杂度的分区、FPGA 硬件上 DSP 有效运行的映射、一个合适存储器架构的发展和在吞吐量、尺寸和能量方面设计目标的实现。但是，读者理解用 FPGA 实现 DSP 功能的细节是很有必要的，这是为了在系统分区和电路架构的发展阶段上得到正确的推断。

比如在系统的分区上，很明显，目前正在研究的系统消耗的资源将超过专用乘法器的可利用资源。然后设计者会面临许多的选择，为了通过使用 LUT 和专用加法器来创造额外的乘法器，要么限制设计空间来使设计映射确保仅仅专用乘法器的资源被使用，要么将映射设计到目前的 FPGA 资源上。此外，实际上，附加的乘法运算也许会被固定系数，这意味着需要较少 LUT 的常系数乘法器可能会得到使用。

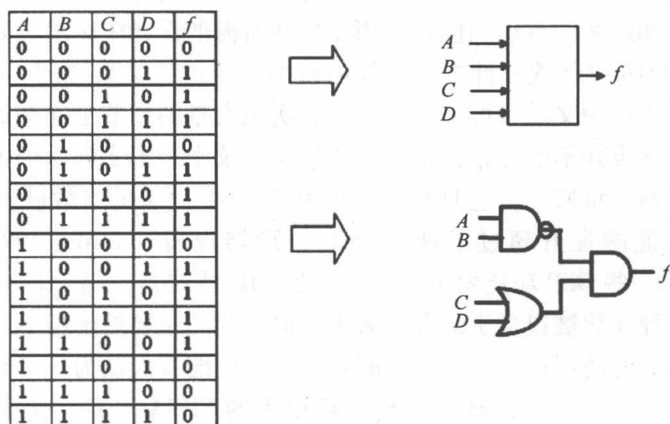
从一个电路架构的角度看，关键目标是为了实现得更快，在这个实现中需要好好权衡尺寸和速率。意识到优化可用在生产所需的设计中是很必要的，因为这可能会决定 FPGA 硬件或软件之间设计的权衡，这也许是系统的一个关键决策。很明显，从发表过的文章中可以看出，除非实施大量的细节工作来调查 FPGA 实现，否则这一决策并不明显，但是，这明显在系统中并不实际，因为设计细节会受到严格的限制。事实上，它是在早期的基于 LUT 设备的有效映射，比如第一个导致实现性能的提升超过了现存 DSP 处理器设备（Goslin 1995）的 Xilinx XC4000，这为一些 DSP 功能做了示范，比如 FIR 滤波器。随着现今的综合工具出现，许多这种优化被包含于综合工具中，但重要的是要理解根本原则。

最后，设计的使用也许会是一个问题：确实可能有人认为一个经济的设计就应该这样，否则设计者会选择一个比需要更大的 FPGA 设备，这会导致成本增加！在这种情况下，实现设计中最后几纳秒定时的最优化也许是很重要的方面。当然，大多数的优化现在逐渐地被隐藏在了综合工具中。

一个简单的例子是运行电路的设计。在一个典型的设计环境中，会描述这个设计为一个有限状态机（Finite State Machine，FSM），每个状态将有一个高位。综合工具将此视为一个热编码的 FSM 机器，但很显然，通过编写一个寄存器链的代码，可以通过使用预设其中一个触发器为“1”来实现。这从FPGA有一套预设触发器的角度来理解。合成工具可能会生成一个高效设计，但明确的是，合成工具可以通过编码电路中的关键部分，以直接的方式来删除这个高效的设计。当然，这可以被视为违反编写代码尽可能简洁的设计原则。因此，本章的重点是详细介绍一些合理的细节，这些技术具体的设计优化，并显示功能是如何映射到 FPGA 结构的。特别是，本章将着眼于 FPGA 存储的实现，这往往是许多应用中的一个关键问题。此外，也包括采用先进的设计技术在 DSP 中的应用，如 DA 和复用技术用于建构系数特定的 DSP 功能。

6.2 LUT 的各种形式

大多数 FPGA 供应商选择 LUT 作为逻辑基本构件的原因是：一个 n 输入的 LUT 能实现任何 n 输入的逻辑功能。如图 6-1 所示，与逻辑门相比的 LUT 的使用在设计上有不同的限制。比如，框图显示功能是如何被映射到逻辑门中的，其中门的数量是有设计限制的。从 LUT 的实现角度来看这并不重要，但是，根本标准在决定 LUT 的数量时直接与输入的数量相关，以及有时候与设计中输出的数量相关，而与逻辑复杂度无关。对于早期的 Xilinx 设备，一个有能力有效扩展到 5 输入的 4 输入 LUT 是核心单元，但是在最新的 Xilinx Virtex™-5 FPGA 技术中，这已经能被扩展到 6 输入的 LUT，并且在最新的 Altera Stratix® Ⅲ FPGA 设备



A	B	C	D	f
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

图 6-1　映射逻辑功能到 LUT 中

中已经被扩展到 8 输入 LUT。

但这仅仅是使用一个 n 输入 LUT 作为 FPGA 的主要逻辑模块的其中一个原因。图 6-2 所示的 Xilinx 公司的 CLB 资源 FPGA 设备，包括了左边的两个 LUT、右边的两个触发器及中间快速进位加法链。这个图突出了左边的 LUT 资源怎么能不仅被用作一个 LUT，也能作为运行一个可编程移位的移位寄存器及一个 RAM 存储单元。

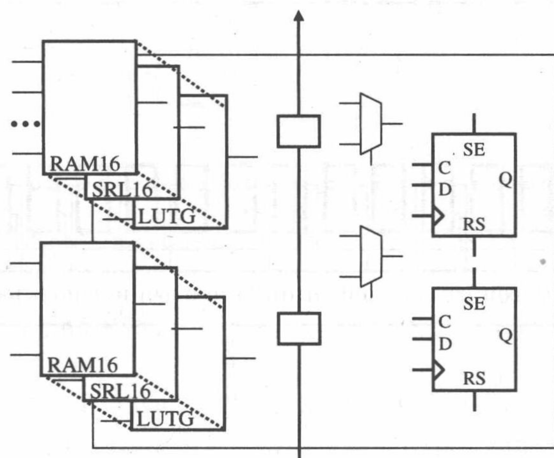


图 6-2　CLB LUT 资源的多种功能

LUT 被用作一个移位寄存器的基本原则在 Xilinx 应用手册（Xilinx Inc. 2005）中有详细的介绍。第一，LUT 可以被认为是一个如图 6-3 所示的 16:1 数据选择器，其中输入地址（这里一个 4 位输入对应一个 Xilinx Spartan 设备）被用于在 RAM 中具体输入的存储地址。在这种情况下，数据选择器会被当作已经是固定的数据选择器。例如，Xilinx 的 16 位移位寄存器 SRL16，配置固定 LUT 值来代替作为一个可寻址移位寄存器，如图 6-3b 所示。

移位寄存器输入同 Xilinx Inc.（2005）中给出的 LUT 同步 RAM 配置一样，也就是一个数据输入、时钟和时钟使能。LUT 使用一个在 Xilinx 库中叫作 Q15 的特定输出，这实际上是由最后的触发器提供的输出。

设计过程如下。

设定一个地址 0111，存储坐标的值作为一个输出被读出，同时，一个新值被读入，被认为是一个新的输入，如图 6-3b 所示的 DIN。如果下一个地址是 0000 并且地址值递增，则直到下一个 0111 地址会花费 8 个周期，与 8 移位时延一致。用这个方法，地址的大小就能够模仿移位寄存器的时延大小。因此并不会

像移位寄存器一样移动所有数据，而是将数据静态地存储在一个 RAM 中，并且变化的地址线通过在正确的时间读出相关数据，来模仿移位寄存器的结果。

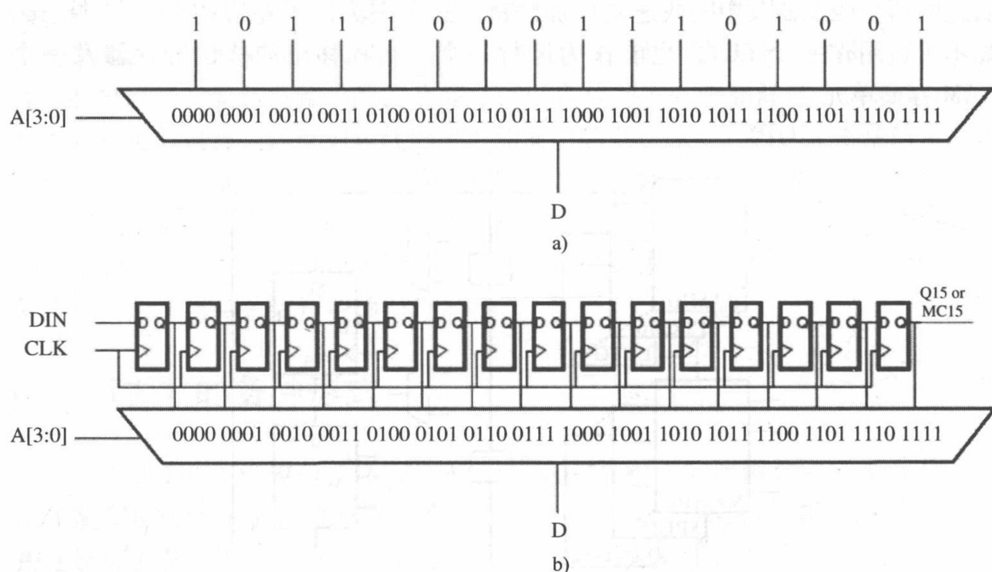


图 6-3 Xilinx CLB LUT 的配置（Xilinx Inc. 2005）（由 Xilinx Inc. 许可复制）
a）16:1 数据选择器 b）移位寄存器

逻辑单元 SRL 结构的细节如图 6-4 所示，图中指的是有一个 4 输入 LUT 的 Xilinx Spartan FPGA 系列。这个单元有一个相关的触发器和一个数据选择器，二者构成整个单元。触发器使用时钟来提供一个同步写入的功能，并且附加的数据选择器可允许一个 DI 直接输入，或者如果一个大的移位寄存器可以被实现，则会允许来自上面单元的一个 SHIFTIN 输入。地址线能被动态使用，但是在同步设计实现中，可以设想它们会同步于时钟。

这对 DSP 系统的实现有巨大启示。正如第 8 章中的演示，均匀性 DSP 运算通过使用硬件共享来减少电路面积。实际上，这在原始电路中会导致一定的时延，如果这个转换会导致存储的巨大提升，那么所有降低复杂度的关键方法都已经失效了。能够将 LUT 当作除了目前触发器以外的移位寄存器来使用，将得到一个高效的实现。因此，为了实现 CLB 的最佳利用，系统级上的一个关键设计标准是为了平衡触发器和 LUT 资源。这在书中后面的大量设计中可以看到。

图 6-4 详细的 SRL 逻辑结构（Xilinx Inc. 2005）（由 Xilinx Inc. 许可复制）

6.3 可用的几种存储器

FPGA 提供许多不同类型的存储器，从模块式 RAM 到可用的多重 LUT 形式的高度分布式 RAM，再到在 FPGA 构造中得到广泛利用的触发器中的数据存储。正如先前章节所讲述的，LUT 和触发器存储能力的交换能得到实施，但这只是对于小分布式的存储器。也许会有需要存储大量输入数据的情况，比如在图像处理应用中，图像的数据模块，或者大量的系数数据，比如在一些 DSP 应用中，特别是已经使用运算的多路复用的应用。在这些情况下，必要的要求是一个较大的模块式 RAM。

FPGA 系列现在采用非常大的板载 RAM，见表 6-1。此表给出了来自 Altera 和 Xilinx 的，类似 DSP 的 FPGA 设备的细节。同时考虑到供应商高端系列的 Xilinx Virtex™-5 和 Altera Stratix® FPGA 技术，可以总结出模块式 RAM 已经从在 FPGA 电路中占较小比例发展到占整个电路的 1/15 或 1/10。典型的，这些比率对于逻辑偏向的 FPGA 系列趋于更低。除了较小分布式存储器以外，FPGA 系列也控制更大的模块式 RAM，这个更大的模块有着一个优势，那就是它们是双端口的，这为一些 DSP 应用提供灵活性。Virtex-5 模块式 RAM 可存储 36kbit 的数据，并且能被配置为两个独立的 18kbit RAM，或者配置为一个 36kbit RAM。每个 36kbit 模块式 RAM 能被配置为一个 64k ×1（当与一个邻近的 36kb 模块式 RAM 串联时），32k ×1，16k ×2，8k ×4，4k ×9，2k ×18 或 1k ×36 存储器。每个 18kb 模块式 RAM 能被配置为一个 16k ×1，8k ×2，4k ×4，2k ×9 或1k ×18 存储器。

表 6-1 Xilinx Virtex 4 和 5 及 Spartan FPGA 和 Altera Stratix 及 Cyclone FPGA 的 FPGA RAM 尺寸的比较

系列	类型	模块式 RAM（kbit）	分布式 RAM（kbit）	Mults	I/O	BRAM/LUT
Virtex	XC5VSX35T	3024	520	192	360	5.8
	XC5VSX95T	8784	1520	640	640	5.8
	XC4VSX25	2304	160	128	320	14.4
	XC4VSX55	5760	384	320	640	15.0
Spartan	XC3S100E	72	15	4	108	4.8
	XC3S1600E	648	231	36	376	2.8
Stratix	EP3SE50	5328	594	384	480	9.0
	EP3SE260	14688	3180	768	960	4.6
Cyclone	EP2C5	117	72	13	158	1.6
	EP2C70	1125	1069	150	622	1.1

本节已经突出了在两个一般的 FPGA 系列中存储器的存储范围。这提供了一个清晰的机制来开发一个存储器层次结构来适应广泛的 DSP 应用。在图像处理应用中，系统的不同部分需要不同尺寸的存储器。比如，图 6-5 所示的快速运动估计电路，其目的是为了在专用硬件上运行高度复杂的运动估计（Motion Estimation，ME）功能。图 6-5 显示了分级存储器系统及实现这种系统的数据带宽考虑。为了运行 ME 功能，有必要下载当前块（Current Block，CB）数据和尺寸，也就是检索窗口（Search Window，SW）到本地存储器。考虑到典型的 SW 是 24 × 24 像素，CB 尺寸是 8 × 8 像素并且一个像素是 8bit，这与典型的存储在嵌入式模块式 RAM 中 3k 和 0.5k 存储器文件一致。这是因为嵌入式 RAM 对于存储这种数据是非常有效的机制，而且带宽速率也并不高。

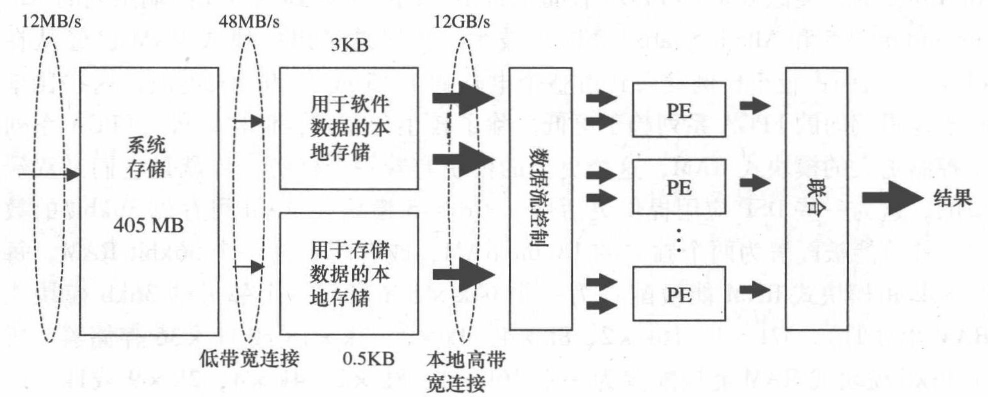


图 6-5 运动估计 IP 核的数据流

在一个 FPGA 实现中，也许有必要实现大量的硬件模块或 IP 核，用来执行 ME 运算，因此也许会要求执行大量的并行计算。这需要使用较小的内存，它可以是较小的分布式 RAM，如果有需要，则也可以是基于 LUT 的存储器和触发器。但是，问题并不仅仅是数据存储，还有如表 6-1 中阐述的较高数据速率。更小的分布式 RAM、基于 LUT 的存储器和触发器，当它们拥有各自的接口时，能提供很高的数据速率。这个数据速率可与更大的 RAM 相媲美，事实上每个存储器的写或读都能够并行执行，因此会得到很高的数据速率。很明显，甚至在一个具体应用中，对存储器大小和数据速率都有明显不同的要求，使用不同类型和尺寸的存储器，在 FPGA 中是很重要的。

6.4　固定系数设计技术

通常需要一个完全可编程数据选择器，这也是 FPGA 供应商现在已经拥有相当数量的 FPGA 实现的原因。但是在一些 DSP 应用中，有时候需要通过一个单一的系数值执行一个字的乘法，比如在一个固定滤波器运算中，或者比如 DCT、FFT、DSP 变换。在一个处理器实现中，这几乎没有作用，但是在专用硬件中，有机会改变执行任务所需的硬件复杂度，因此，专用系数倍率或固定系数倍率（Fixed Coefficient Multiplication，KCM）有可能减少电路过载。大量机制已经被用于派生 KCM。这些包括 DA（Goslin 和 Newgard 1994）、字符串编码和消元（Cocke 1970，Feher 1993）。

这个观点很好地被应用到了 FPGA 设备中，其中只有 LUT 和专用快速加法器可被用于乘法器，因此在实现这一系列定值系数功能中获得了一块可用面积（Goslin 和 Newgard 1994，Peled 和 Lin 1974）。大量的技术已经演变成了定值系数的乘法，而大量的 FPGA 架构以专用乘法器或 DSP 模块的形式来创建专用的硬件板载乘法器，仍然是值得回顾可借鉴的方法。本节考虑了被用于单一定值系数的乘法（Peled 和 Liu 1974）的 DA 使用并且减少了能够乘以一系列系数值（Turner 和 Woods 2004）的系数乘法器（Redued Coefficient Multiplication，RCM）方法。

6.5　分布式体系结构

DA 是执行乘加运算的一个有效技术，其中乘法被重构，以便乘法和加法同时在数据和单系数位上得到执行。这项技术的原则是基于假设我们会储存计算的值而不是执行计算（FPGA 的 LUT 已经提供了这个功能）。

假设在式（6-1）中进行乘积和的计算，其中 x_j 代表一个数据流并且 a_0，a_1，…，a_{N-1}代表一系列系数值。我们能使用 LUT 来生成数据并且使用快速加

法器来计算最后的乘法，而不是通过与门来计算部分结果。一个 8 输入的基于 LUT 的乘法器例子如图 6-6 所示，可以看出这个乘法器要求一个相当大的 4kbit LUT 存储器资源。

$$y = \sum_{i=0}^{N-1} a_i x_i \tag{6-1}$$

当系数固定时，对存储器的要求会大大降低（例如从 512 位到 8 位），这使 FPGA 架构有一定的吸引力。一种典型的实现是使用一个如图 6-6 所示的基于 LUT 的乘法器电路来执行乘法 a_0x_0。但是，可通过使用 DA 产生一个更加有效的结构，在 Peled 和 Liu（1974），Meyer. Baese（2001），White（1989）中得到了详细的描述。根据 White（1989）中描述的方法得到下面的分析。在以下的分析中，记下系数 a_0，a_1,. . . ，a_{N-1}，这代表一系列固定系数值。

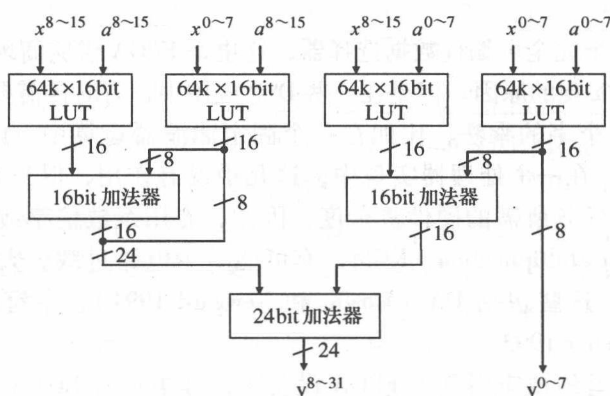


图 6-6 基于 LUT 的 8 输入的乘法器

假设输入数据流由一个二进制补码有符号数 x_n 表示，如式（6-2）。

$$x_n = -x_n^0 + \sum_{j=1}^{M-1} x_n^j 2^j \tag{6-2}$$

式中，$x_n^j 2^j$ 表示数据流 x_n 的第 n 个样值的第 j 位，并且 x_n^0 表示符号位，因此在等式中表示为负数，数据字长为 Mbit。y 的计算能被重写，如式（6-3）。

$$y = \sum_{i=0}^{N-1} a_i \left(-x_n^0 + \sum_{j=1}^{M-1} x_i^j 2^j\right) \tag{6-3}$$

乘法完全扩展可以得到式（6-4）。

$$y = \sum_{i=0}^{N-1} a_i(-x_n^0) + \sum_{i=0}^{N-1} a_i \sum_{j=1}^{M-1} x_i^j 2^j \tag{6-4}$$

整理后得

$$
\begin{aligned}
y &= a_0(-x_0^0) + a_0(x_0^1 2^1 + x_0^1 2^2 + \cdots + x_0^{M-1} 2^{M-1}) \\
&+ a_1(-x_1^0) + a_1(x_1^1 2^1 + x_1^2 2^2 + \cdots + x_1^{M-1} 2^{M-1}) \\
&\vdots \\
&+ a_{N-1}(-x_{N-1}^0) + a_{N-1}(x_{N-1}^1 2^1 + x_{N-1}^2 2^2 + \cdots + x_{N-1}^{M-1} 2^{M-1})
\end{aligned}
$$

整理可得到式（6-5）。

$$
y = \sum_{i=0}^{N-1} a_i(-x_i^0) + \sum_{j=0}^{M-1} \left[\sum_{i=0}^{N-1} a_i(x_i^j) \right] 2^j \tag{6-5}
$$

并且再一次，扩展过后的样子给予了怎么得到如下重组的样子的一个清楚的方法。

$$
\begin{aligned}
y &= 2^0(a_0(-x_0^0) + a_1(-x_1^0) + \cdots + a_1(-x_{N-1}^0)) \\
&+ 2^{-1}(a_0 x_0^1 + a_1 x_1^1 + \cdots + a_{N-1} x_{N-1}^0) \\
&\vdots \\
&+ 2^{-M+1}(a_0 x_0^{M-1} + a_1 x_1^{M-1} + \cdots + a_{N-1} x_{N-1}^{M-1})
\end{aligned}
$$

考虑到系数现在是固定值，式（6-5）中的 $\sum_{i=0}^{N-1} a_i(x_n^i)$ 仅有 2^K个可能取值并且 $\sum_{i=0}^{N-1} a_i(-x_n^0)$ 也仅有 2^K 个可能取值。因此实现能被存储在2×2^K位大小的一个 LUT 中，其中靠前的 8 位可以代表 512 位的存储。

考虑实现中 $N=4$，为了知道它怎样适合基于 LUT 结构的 FPGA，就给出了如式（6-6）扩展后的表达式。

$$
\sum_{i=0}^{N-1} a_i(x_i^j) = a_0(x_0^0) + a_1(x_1^0) + a_2(x_2^0) + a_3(x_3^0) \tag{6-6}
$$

这显示了如果使用输入 x 作为 LUT 的地址，那么存储值见表 6-2。通过重排式（6-7）来实现式（6-8）中的表达式，可以看出，这个计算的 LUT 内容是表 6-2 中储存的倒数，并且能够通过执行二进制补码的减法来得到实施，而不是执行加法。通过围绕一个加法器来轮流计算，这个计算能使用并行硬件执行或顺序执行，如图 6-7 所示，其中计算的最后阶段是一个减法运算而不是加法。很明显能看出，这种计算可以使用 Xilinx 公司的 FPGA 系列的基本 CLB 结构，以及来自 Altera 的 4 位的 LUT 来储存 DA 数据。快速加法器被用于执行加法运算，并且通过使用触发器，数据能得到储存。实际上，一个 CLB 能执行计算的一“位”，意味着现在 8 个 LUT 只需要用比并行结构更慢的速度来进行计算，这都归因于计算的连续性。

表 6-2 DA 计算的 LUT 内容

地址				LUT 内容
x_3^0	x_2^0	x_1^0	x_0^0	
0	0	0	0	0
0	0	0	1	a_0
0	0	1	0	a_1
0	0	1	1	a_1+a_0
0	1	0	0	a_2
0	1	0	1	a_2+a_0
0	1	1	0	a_2+a_1
0	1	1	1	$a_2+a_1+a_0$
1	0	0	0	a_3
1	0	0	1	a_3+a_0
1	0	1	0	a_3+a_1
1	0	1	1	$a_3+a_1+a_0$
1	1	0	0	a_3+a_2
1	1	0	1	$a_3+a_2+a_0$
1	1	1	0	$a_3+a_2+a_1$
1	1	1	1	$a_3+a_2+a_1+a_0$

$$\sum_{i=0}^{N-1} a_i(x_i^0) = a_0(x_0^0) + a_1(x_1^0) + a_2(x_2^0) + a_3(x_3^0) \tag{6-7}$$

$$\sum_{i=0}^{N-1} a_i(-x_i^0) = x_0^0(-a_0) + x_1^0(-a_1) + x_2^0(-a_2) + x_3^0(-a_3) \tag{6-8}$$

很明显看出，这个技术为一系列的其中一部分是固定的 DSP 功能提供了相当大的优势。这包括一些固定的 FIR 滤波器和 IIR 滤波器，一系列的固定变换，也就是 DCT 和 FFT 及其他选出来的计算。这个技术已经在别处得到了详细的介绍（Peled 和 Liu 1974，Meyer-Baese 2001，White 1989），并且一大部分的应用笔记可从 FPGA 供应商的相关主题中得到。

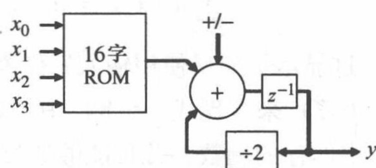


图 6-7 基于 DA 的乘法器模块框架

6.6 折减系数乘法器

DA 方法已经取得了相当大的成功，并且已经成为在 DSP 计算中使用早期 FPGA 技术的重点。但是，此技术的主要限制是为了实现面积的缩小，系数必须

是固定的。如果一些应用要求取所有的系数，则必须使用一个完全的可编程乘法器，这在早期的设备中是很昂贵的，因为它必须是目前 LUT 和加法器资源的结合，虽然在更多最近的 FPGA 系列中已经以 Xilinx Virtex™ -5 和 Altera Stratix® 中的专用 DSP 硬件形式得到提供。但是，在一些比如 DCT 和 FFT 的应用中，需要有限制范围的乘法器。这里仅对 DCT 计算进行了阐述。

DCT 是一个重要的变换，在图像压缩技术中得到了广泛的使用。二维 DCT 通过将一个模块的像素变换为一个存在于模块中的，与空间频谱相关的系数集合而得到运行，如式（6-9）。

$$y(k,l) = \alpha(k)\alpha(l)\sum_{n=0}^{N-1}\sum_{m=0}^{N-1}x(n,m)c(n,k)c(m,k) \tag{6-9}$$

其中，$c(n,k) = \cos[(2n+1)\pi k/2N]$ 且 $c(m,k) = \cos[(2m+1)\pi k/2N]$，并且 k 和 l 的范围是 $0 \sim N-1$；值 $\alpha(k)$ 和 $\alpha(l)$ 是标定变量。典型的，利用了函数的可分离性质，以此来允许其被分解为两个连续的一维变换。这是通过使用比如行列的分解技术，在两个一维变换之间进行矩阵变换函数得以实现的。

一个 N 点一维变换等式，涉及一个输入数据的序列 $\boldsymbol{x}(i)$，$i=0>N-1$ 转变成 $\boldsymbol{Y}(k), k=0>N-1$，式（6-10）和式（6-11）中给出了转换过程。

$$\boldsymbol{Y}(0) = \alpha(0)\sum_{n=0}^{N-1}\boldsymbol{x}(i) \tag{6-10}$$

$$\boldsymbol{Y}(k) = \alpha(k)\sum_{n=0}^{N-1}\boldsymbol{x}(i)\cos\left[\frac{k\alpha(2i+1)}{2N}\right], \forall k = 1,\cdots,N-1 \tag{6-11}$$

式中，$\alpha(0)=1/\sqrt{N}$，此外 $\alpha(k)=2/\sqrt{N}$。将等式扩展为一个矩阵形式，可以得到以下矩阵向量计算。

$$\begin{bmatrix}\boldsymbol{Y}_0\\\boldsymbol{Y}_1\\\boldsymbol{Y}_2\\\boldsymbol{Y}_3\\\boldsymbol{Y}_4\\\boldsymbol{Y}_5\\\boldsymbol{Y}_6\\\boldsymbol{Y}_7\end{bmatrix} = \begin{bmatrix}\boldsymbol{C}_4 & \boldsymbol{C}_4 & \boldsymbol{C}_4 & \boldsymbol{C}_4 & \boldsymbol{C}_4 & \boldsymbol{C}_4 & \boldsymbol{C}_4 & \boldsymbol{C}_4\\ \boldsymbol{C}_1 & \boldsymbol{C}_3 & \boldsymbol{C}_5 & \boldsymbol{C}_7 & -\boldsymbol{C}_7 & -\boldsymbol{C}_5 & -\boldsymbol{C}_3 & -\boldsymbol{C}_1\\ \boldsymbol{C}_2 & \boldsymbol{C}_6 & -\boldsymbol{C}_6 & -\boldsymbol{C}_2 & -\boldsymbol{C}_2 & -\boldsymbol{C}_6 & \boldsymbol{C}_6 & \boldsymbol{C}_2\\ \boldsymbol{C}_3 & -\boldsymbol{C}_7 & -\boldsymbol{C}_1 & -\boldsymbol{C}_5 & \boldsymbol{C}_5 & \boldsymbol{C}_1 & \boldsymbol{C}_7 & -\boldsymbol{C}_3\\ \boldsymbol{C}_4 & -\boldsymbol{C}_4 & -\boldsymbol{C}_4 & \boldsymbol{C}_4 & \boldsymbol{C}_4 & -\boldsymbol{C}_4 & -\boldsymbol{C}_4 & \boldsymbol{C}_4\\ \boldsymbol{C}_5 & -\boldsymbol{C}_1 & \boldsymbol{C}_7 & \boldsymbol{C}_3 & -\boldsymbol{C}_3 & -\boldsymbol{C}_7 & \boldsymbol{C}_1 & -\boldsymbol{C}_5\\ \boldsymbol{C}_6 & -\boldsymbol{C}_2 & \boldsymbol{C}_2 & -\boldsymbol{C}_6 & -\boldsymbol{C}_6 & \boldsymbol{C}_2 & -\boldsymbol{C}_2 & \boldsymbol{C}_6\\ \boldsymbol{C}_7 & -\boldsymbol{C}_3 & \boldsymbol{C}_3 & -\boldsymbol{C}_1 & \boldsymbol{C}_1 & -\boldsymbol{C}_1 & \boldsymbol{C}_5 & -\boldsymbol{C}_7\end{bmatrix}\begin{bmatrix}\boldsymbol{X}_0\\\boldsymbol{X}_1\\\boldsymbol{X}_2\\\boldsymbol{X}_3\\\boldsymbol{X}_4\\\boldsymbol{X}_5\\\boldsymbol{X}_6\\\boldsymbol{X}_7\end{bmatrix} \tag{6-12}$$

在其目前的形式中，矩阵向量的计算要求 64 个乘法及 63 个加法来计算 $\boldsymbol{Y}$ 向量。但是，大量的研究工作已经通过输入数据的预计算来降低 DCT 的计算复杂度，以此来减少乘法的数量。这样一个由 Chen 等（1977）提出的方法实现了式

(6-13）给出的形式。这些类型的优化是可能的，并且一系列这样的变换同时为一维类型（Hou 1987，Lee 1984，M－T－Sun 等 1989）及直接二维实现的 DCT 而存在（Duhamel 等 1990，Feig 和 Winograd 1992，Haque 1985，Vetterli 1985)。

$$\begin{bmatrix} \boldsymbol{Y}_0 \\ \boldsymbol{Y}_2 \\ \boldsymbol{Y}_4 \\ \boldsymbol{Y}_6 \\ \boldsymbol{Y}_1 \\ \boldsymbol{Y}_3 \\ \boldsymbol{Y}_5 \\ \boldsymbol{Y}_7 \end{bmatrix} = \begin{bmatrix} \boldsymbol{C}_4 & \boldsymbol{C}_4 & \boldsymbol{C}_4 & \boldsymbol{C}_4 & 0 & 0 & 0 & 0 \\ \boldsymbol{C}_2 & \boldsymbol{C}_6 & -\boldsymbol{C}_6 & -\boldsymbol{C}_2 & 0 & 0 & 0 & 0 \\ \boldsymbol{C}_4 & -\boldsymbol{C}_4 & -\boldsymbol{C}_4 & \boldsymbol{C}_4 & 0 & 0 & 0 & 0 \\ \boldsymbol{C}_6 & -\boldsymbol{C}_2 & \boldsymbol{C}_2 & -\boldsymbol{C}_6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \boldsymbol{C}_1 & \boldsymbol{C}_3 & \boldsymbol{C}_5 & \boldsymbol{C}_7 \\ 0 & 0 & 0 & 0 & \boldsymbol{C}_3 & -\boldsymbol{C}_7 & -\boldsymbol{C}_1 & -\boldsymbol{C}_5 \\ 0 & 0 & 0 & 0 & \boldsymbol{C}_5 & -\boldsymbol{C}_1 & \boldsymbol{C}_7 & \boldsymbol{C}_3 \\ 0 & 0 & 0 & 0 & \boldsymbol{C}_7 & -\boldsymbol{C}_5 & \boldsymbol{C}_3 & -\boldsymbol{C}_1 \end{bmatrix} \begin{bmatrix} \boldsymbol{X}_0+\boldsymbol{X}_7 \\ \boldsymbol{X}_1+\boldsymbol{X}_6 \\ \boldsymbol{X}_2+\boldsymbol{X}_5 \\ \boldsymbol{X}_3+\boldsymbol{X}_4 \\ \boldsymbol{X}_0-\boldsymbol{X}_7 \\ \boldsymbol{X}_1-\boldsymbol{X}_6 \\ \boldsymbol{X}_2-\boldsymbol{X}_5 \\ \boldsymbol{X}_3-\boldsymbol{X}_4 \end{bmatrix} \tag{6-13}$$

式（6-13）可在一个电路中实现，在这个电路中一个完全的可编程乘法器和加法器的结合能被用于实现等式的一行或一列，或者可能随着一个完整的电路得到实现。但是，不幸的是乘法器在被乘数上只能使用 4 个独立值，如图 6-8 所示。这显示了理想情况下，我们需要一个能处理 3 个独立值的乘法器。另一选择是使用 DA 方法，其中任意 32 个独立 DA 乘法器能被用于实现矩阵计算。还有一个选择是使用 8 个 DA 乘法器来实现硬件的减少，但是数据流会比较复杂，这都是为了在正确的时间加载正确的数据。拥有 DA 乘法复合乘法器是有吸引力的，这将乘以有限范围内的乘数，从而将硬件复杂度与计算的所需条件做交换，这正是 Turner 和 Woods（2004）开发的 RCM 乘法器实现的。

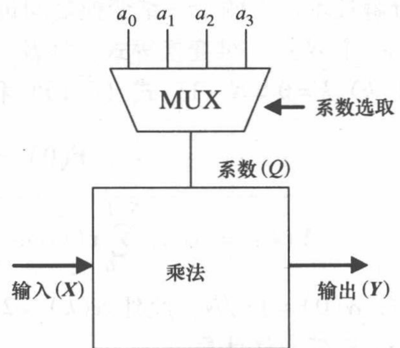


图 6-8　降低了复杂度的乘法器

6.6.1　RCM 的设计过程

先前有关章节突出了 DA 的功能性如何能被映射到一个基于 LUT 的 FPGA 技术中。实际上，如果将乘法器看作一个产生积的结构并使用一个加法器树来求和，以此产生一个最后的输出，那么拥有固定系数及如 DA 中提议的组织计算，将影响映射 LUT 中的乘积项和加法器树的高度功能性。本质上，这是实现了主要的面积上的获利。

在图 6-9 中该观点得到了阐述，虽然这只是浅层次的阐述，但因为实际

"与"和加法运算不是在硬件中被执行的，所以会预计算"与"和加法运算。但是，这给予我们如何能映射额外功能性到此方法核心部分的基于 LUT 的 FPGA 上一个新视角。

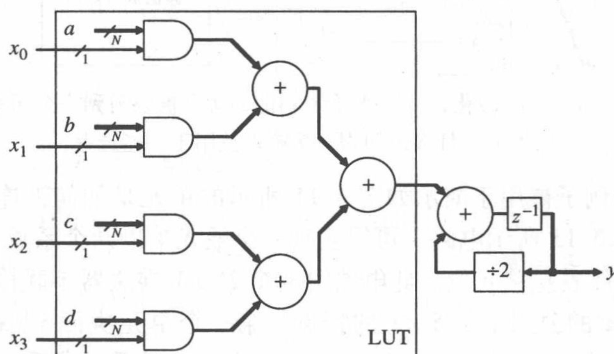


图 6-9　基于 DA 的乘法器模块框图

在 DA 实现中，主要是映射尽可能多的加法树到 LUT 中。如果我们考虑将一个常系数乘法器映射到同样容量的 CLB 中，那么主要的要求是映射快速进位逻辑的 EXOR 功能到如图 6-10 所示的 LUT 中。这不会和 DA 实现一样有效，但是现在多余的输入能被用于实现其他功能，如图 6-11 所示。这是我们创建实现电路过剩结构的出发点（以便实现附加功能）。

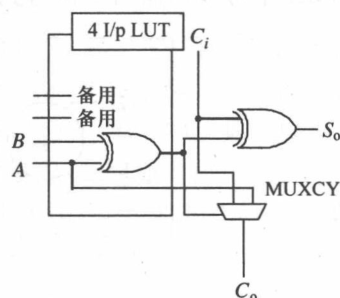


图 6-10　映射乘法器功能到 Virtex 4 CLB 中

图 6-11a 实现 $A+B$ 或 $A+C$ 的功能；图 6-11b实现 $A-B$ 或 $A-C$ 的功能；图 6-11c 实现 $A+B$ 或 $A-C$ 的功能。这就产生了如图 6-12 所示的一般性结构的观点，并且 Turner（2002）给出了一个自动产生这些结构的详细处理方法。在这里我们使用一个直观的方法来显示通过使用它的技术来映射 DCT 的功能性，但是要注意这不是在 Turner（2002）中使用的技术。

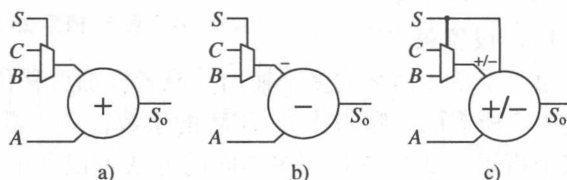


图 6-11　通过使用基于乘法器的设计技术可能的实现

a）第 1 个单元　b）第 2 个单元　c）第 3 个单元

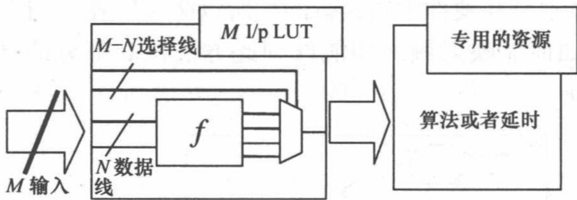


图 6-12 技术的一般化，其中 N 输入功能的集合被映射到一个 M 输入的 LUT 上并且多余的 $M-N$ 输入被用于功能选择

一些简单的例子被用于演示如图 6-11 所示的单元是如何被连接在一起来构建乘法器的。图 6-13 所示电路，可以实现一个输入乘以两个系数，也就是 45 或 15。符号 $x\times 2^n$ 代表左移 n 位。此电路由两个 $2^n\pm 1$ 乘法器串联构成，主要利用了$45\times x$ 和 $15\times x$ 的公因子是 $5\times x$ 的特点。第一个单元执行 $5\times x$，然后第二个单元通过添加位移 2（2^1）或 8（2^3）来执行进一步的乘 9 或乘 3 运算，这要依靠乘法器控制信号的设置（15/45 标志）。移位运算不要求任何硬件支持，因为在 FPGA 中可以像路由一样得到实现。

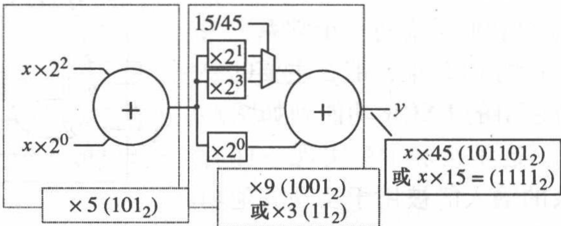


图 6-13 通过 45 或 15 的乘法器

图 6-14 所示为乘以 45 或乘以素数 23 的一个电路。在这里不能使用公因子，而是使用对 45 进行不同的因子分解，与三步操作相比，仅一步操作就可得到 15，使用减法来产生 15 这个乘法，也就是（16 − 1）。设置第二个单元，来添加从第一个单元而来的多个输出，或者添加可移位的输入 x。使所得的乘法器实现这两个所需的系数来作为 KCM 的系数，而不需要重新配置。另外，此例给出了某个指示，那就是有许多不同的方法可以映射出想要的系数集合，并排列单元以便获得一个有效的乘法器结构。

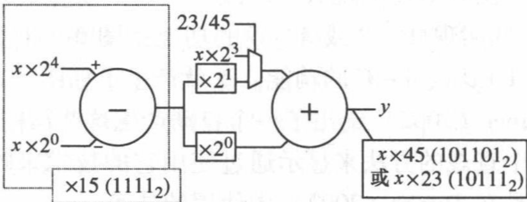


图 6-14 乘 45 或乘 23 的乘法

在 Turner 和 Wood（2004）中，作者已经推导出了可实现最佳解决方案的方

法，这是对考虑之中的特殊 FPGA 结构而言的。第一步涉及识别图 6-11 中所示的所有单元类型。这些仅仅是显示 Xilinx Virtex™-Ⅱ系列的一个小样本。单元的范围取决于 LUT 输入的数量及专用硬件资源。下一步是进行系数编码，以确定最有效的结构，这证明是有符号的数字（标准差）编码。因此这些系数得到编码并产生移位的有符号数字（Shifted Signed Digit，SSD），然后被分组来发展最后 RCM 电路的树结构。原型 C ++ 软件得到了发展来使这个过程自动化。

6.6.2 FPGA 的乘法器综述

DA 技术已经被证明，可以在固定系数功能的应用中良好运行。正如在第 2 章中看到的，这不仅仅对固定系数的运算有限制，比如固定系数的 FIR 滤波器和 IIR 滤波器，而且在如 DCT 和 FFT 的固定变换中也有应用。但是后者为 RCM 设计技术提供了一个更好的解决方法，因为主要的要求是发展乘以一个范围内系数的乘法器，而并不是乘以一个单一值。RCM 技术已经在 DCT 及一个多相滤波器中做了演示，这个滤波器在性能方面，与其他基于 DA 技术实现的定值 DSP 功能的质量相当（Turner 和 Woods 2004，Turner 等 2002）。

必须说明 FPGA 架构中的变化，主要发展 Xilinx Virtex™-5 FPGA 技术中的 DSP48s 和最新的 Altera Stratix ® ⅢFPGA 设备中的 DSP 功能模块，它们都已经减小了构建固定系数甚至是限制系数的范围结构，将它们作为专用的乘法器为基础的硬件模块提供更优越的性能。然而，在可能的情况下，FPGA 资源是有限的，并且将这些技术配合流水线使用，会导致这些专用模块用同样速率来实现性能。因此，要求了解这些目前的技术是很有用的。

6.7 总结

本章目的在于介绍一些技术，特别是着眼于映射到特定 FPGA 平台的 DSP 系统的技术。许多人会认为，在改进技术及缩小生产率差距的这段时期，我们应该完全远离设计方法这个方面。虽然这种观点是好的意向，但是在许多的情况下，详细的实现方法在实现实际的电路中变得更重要了。

如图 6-5 所示的设计例子中的图像处理实现，一个合适的硬件架构的派生是在速度与尺寸方面，理解什么是在根本资源的基础上派生的。因此对实际 FPGA 限制有清晰的理解在发展一个合适的架构中是很重要的。一些其他的固定系数技术也许在应用中有用，但在这些应用中硬件是被限制的，并且也许使用者希望可以在应用的其他部分的 DSP 资源之间做交换。虽然没有在本章中特别介绍，但是高效的 FPGA 实现的一个关键方面是高效的设计风格的发展。一个好的处理方法由 Michael Keating 和 Pierre Bricaud 在他们的 Reuse Methodology Manual for System - On - A - Chip Designs（Keating 和 Bricaud 1998）中给出。该文章的主题是突出大

量优秀的设计风格，这些设计风格应该被有效的 HDL 代码创建，在 SoC 平台上实现。除了好的 HDL 编码指示外，本文也提供了混合时钟边沿的一些关键的建议，发展了复位和使能电路及门控时钟的方法。这些是很有必要的，但是有读者认为，这本书的视野集中在一个高层次电路架构的产生上。

参考文献

Chen WA, Harrison C and Fralick SC (1977) A fast computational algorithm for the discrete cosine transform. *IEEE Trans on Comms.* COM-25(9), 1004–1011.

Cocke J (1970) Global common subexpression elimination. *Proc. Symp. on Compiler Construction*, pp. 850–856.

Duhamel P, Guillemot C and Carlach J (1990) A DCT chip based on a new structured and computationally efficient DCT algorithm. *Proc. IEEE Int. Symp. on Circuits and Systems*, pp. 77–80.

Feher B (1993) Efficient synthesis of distributed vector multipliers. *Euromicro Symp. on Microprocessing and Microprogramming*, pp. 345–350.

Feig E and Winograd S (1992) Fast algorithms for the discrete cosine transform. *IEEE Trans. on Signal Processing* 40(9), 2174–2193.

Goslin G (1995) Using xilinx FPGAs to design custom digital signal processing devices. *Proc. DSPX*, pp. 565–604.

Goslin G and Newgard B (1994) 16-tap, 8-bit FIR filter applications guide. Web publication downloadable from www.xiLinx.com.

Haque MA (1985) A two-dimensional fast cosine transform. *IEEE Trans. on Acoustics, Speech, and Signal Processing* 33(6), 1532–1539.

Hou HS (1987) A fast recursive algorithm for computing the discrete cosine transform. *IEEE Trans. Acoustics, Speech, and Signal Processing* 35(10), 1455–1461.

IRTS (1999) *International Technology* Roadmap for Semiconductors, 1999 edn. Semiconductor Industry Association. http://public.itrs.net

Keating M and Bricaud P (1998) *Reuse Methodology Manual for System-On-A-Chip Designs*. Kluwers, Norwell, MA, USA.

Lee BG (1984) A new algorithm to compute the discrete cosine transform. *IEEE Trans. on Acoustics, Speech and Signal Processing* 32, 1243–1245.

Meyer-Baese U (2001) *Digital Signal Processing with Field Programmable Gate Arrays*. Springer, Germany.

Sun MT, Chen TC and Gottlieb AM (1989) VLSI implementation of a 16x16 discrete cosine transform. *IEEE Transaction on Circuits and Systems* 36, 610–617.

Omondi AR (1994) *Computer Arithmetic Systems*. Prentice Hall, New York.

Peled A and Liu B (1974) A new hardware realisation of digital filters. *IEEE Trans. on Acoustics, Speech and Signal Processing*, pp. 456–462.

Turner RH (2002) *Functionally diverse programmable logic implementations of digital processing algorithms*. PhD dissertation, Queen's University Belfast.

Turner RH and Woods R (2004) Highly efficient, limited range multipliers for LUT-based FPGA architectures. *IEEE Trans. on VLSI Systems* 12, 1113–1118.

Turner RH, Woods R and Courtney T (2002) Multiplier-less realization of a poly-phase filter using LUT-based FPGAs *Proc. Int. Conf. on Field Programmable Logic and Application*, pp. 192–201.

Vetterli M (1985) Fast 2-d discrete cosine transform. *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, pp. 1538–1541.

White SA (1989) Applications of distributed arithmetic to digital siganl processing. *IEEE ASSP Magazine*, pp. 4–19.

Xilinx Inc. (2005) Using look-up tables as shift registers (srl-16) in spartan-3 generation FPGAs. Web publication downloadable from http://www.xilinx.com.

第7章　FPGA的快速DSP系统设计工具和流程

7.1　引言

由硅集成技术中不间断成长激励的计算架构的革新已经出现了大量新的DSP实现技术的发展。这些平台包括单芯片多处理器或多样化SoC的解决方案及FPGA。正如第1章所突出的，对这种技术中计算架构的革新已经有许多年了，通过使用它们，已经超过了设计者独自实现DSP系统的能力。这个观点已经被流行地称作设计生产率差距（IRTS 1999），并且产生它的主要原因在工业向实现SoC设计流程驱动中是一个主要限制因素。这推动电子设计自动化工业重新考虑系统设计的观念（Keutzer等2000，Lee等2003，Rowson和Sangiovanni Vincentelli 1997）。

现代的DSP实现平台由多样化的处理架构混合组成，包括微控制器单元（Microcontroller Unit，MCU）冯·诺依曼似的处理架构，增加计算能力的处理器，比如VLIW DSP微处理器，或者为有效实现任务的专用硬件。现代FPGA的革新意味着它是作为一个实现平台的潜在候补，要么是作为一个独立的SoPC，要么是作为一个对现有基于软件平台的补充。基于FPGA的嵌入式平台提出全新的并且更加复杂的实现发表给设计者，这都是归因于缺少一个有定义的处理架构。

这个有着广泛目标的处理架构和相应的实现技术使得DSP系统在目前的设计抽象阶段得以实现是一个很费力的任务。因此，连贯迅速的实现框架的使用是很关键的，这种框架直接将一个行为系统转化成一个嵌入式的表现形式。本章概括了目前对这个问题的解决方法。

一般的，基于现代嵌入式系统的设计观念在迅速地成长普及。这个观念是对大量具体设计方法论类型和工具的概括，它为了算法行为的表达促进了语义定义良好建模语言的使用。每个模型类型的语义都得到开发来提供迅速的实现能力。正如这些，对任何的设计方法，它们都有两个重要的方面，即因为算法规范使用的特殊计算模型（Model of Computation，MoC），以及用于引向一个实现这些模型的细化的方法学。本章将概述特别是在这些领域的主要发展。

从这之后，FPGA专门的系统综合有许多问题。有三点在目前的可编程逻辑

体系中是最突出的，即专用硬件 IP 核和核心网络的综合；对多重处理器架构的软件综合；以及最后的为了引导和自动实现一个在分层 FPGA 架构上给出的应用的大规模系统级设计。这三个方面在每个领域中有大量的典型设计工具来处理。

本章由以下几部分组成。对某些设计方法和工具流行和需要的原因与 FPGA 设备的革新高度地连接起来，并且这个动机将在 7.2 节中会有概述。以上概述的使能工具集的基础设计方法和建模语言将在 7.3 节中会有介绍。7.4.3 节中将介绍这种模型化和迅速实现技术是如何为单个和多处理器软件综合得到开发利用的。对于 IP 核和核心网络综合的工具和技术将在 7.5 节中会有介绍。7.6 节将集中介绍自动生成的多样化 FPGA 架构及映射算法规范到这些架构上的工具。

7.2 FPGA 系统设计的革新

自从第一台设备出现以来，由 FPGA 制订的非传统革新途径已经强烈地影响了设计方法和现代设备的工具要求。我们认为有三个明显的“FPGA 的时代”，它们每一个都拥有自己独特的设备设计和编程方法。

7.2.1 时代一：定制胶合逻辑

当 FPGA 第一次出现时，在单个设备上逻辑资源的相对缺乏（通过今天的标准）意味着极少的实质性功能能在单个芯片上实现，但是它们的编程本质意味着 FPGA 成为了多芯片 ASIC 处理平台的可定制胶合逻辑理想的主设备。低等级的复杂度意味着使用芯片架构和功能性的门级基于原理图的捕获能用足够高的生产率实现简单芯片的架构。同样的，在那段时期，基于原理图的门级设计是很流行的。

7.2.2 时代二：中密度逻辑

随着新一代的 FPGA 设备的采用，比如 Xilinx XC4000 和 Virtex 设备，以及 Altera Stratix 设备，可实现的逻辑密度级别随着摩尔定律持续提高。再加上并行的潜在例外的高级别，意味着 FPGA 逐渐从一个简单的胶合逻辑使能设备转变成复杂 DSP 组件的主机，比如滤波器和变换式。这个革新使得工业为高性能 DSP IP 核的发展得到提高。这个趋势随着比如来自 Xilinx 的Virtex™和Virtex™ -2 系列的 FPGA 和来自 Altera 的 Cyclone 设备上的乘法器、加法器和存储器高性能小组件的引入持续着。因为这些设备而提高的设计复杂度意味着像 Xilinx ISE 的寄存器传送级（Register Transfer Level，RTL）设计工具和像 VHDL 一样的语言的出现，补充了 RTL 级综合工具的出现来将 EDIF 转换到 FPGA 编程文件中。

这个阶段导致了大量 IP 核的一个遗留问题，以及对与核心网络结构设计工

具相关联的需要已经在目前的FPGA设计方法和工具中持续地扮演着一个重要的角色。这些在7.5节中会考虑到。

7.2.3　时代三：分层级的SoC

Virtex™ -2 Pro和Stratix FPGA设备的出现意味着传统FPGA可编程逻辑是通过比如在Virtex™ -2 Pro中的PowerPC嵌入式微处理器和比如Xilinx Rocket I/O的高速专用串行通信收发器来补充的。从那以来直到目前，FPGA及其性能容量和灵活性已经一起成为单个芯片分级处理的解决方法，这意味着它们能被安置在DSP系统的核心处而不是作为一个附加的加速器。

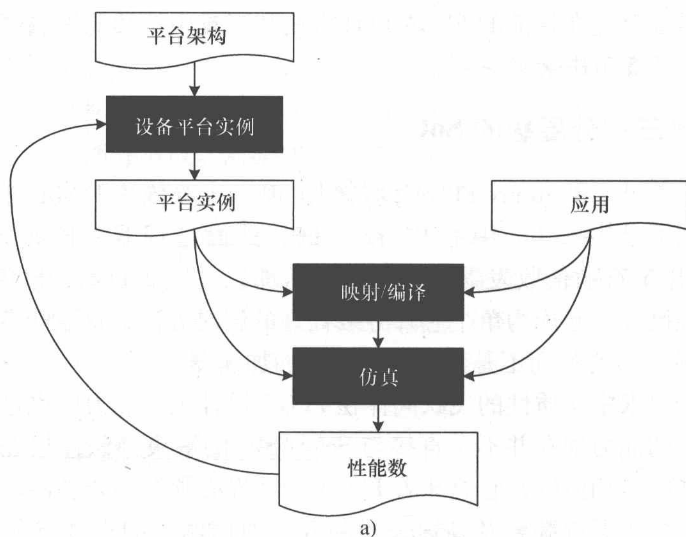
设备性能要求中实质性的飞跃同样在FPGA设计工具能力中也是实质性的进步。这些设备的能力现在并不是直接与SoC ASIC的解决办法连接在一起，同时这些设备的性能是相似的，它意味着FPGA设计界必须在多处理器软件设计、核心网络结构、分级系统综合及高速关闭和开启通信网络的设计方面采用新的技术。不幸的是，这项技术并不容易。

从第一代FPGA到第二代FPGA的演进，是借助于业界内从门级电路到RTL级电路的发展而顺利完成的，而类似的从第二代到第三代演进的推动技术还没有形成。在FPGA中，这种技术应该解决IP核综合与核心网络构建，多处理器系统设计，以及系统集成与优化问题。不幸的是，这些初级的设计工具，比如Xilinx Embedded Development Kit（EDK）和Altera SoPC Builder，仅仅提供这些领域的基本功能。这一章的剩余部分概述了目前最先进的工具和每一个领域中的实践练习，是从全局系统设计方法角度开始的（7.3节），然后将讲解多重处理器软件的综合、IP核和核心网络综合（7.5节），最后7.6节将介绍异构多重处理器架构的设计和综合。

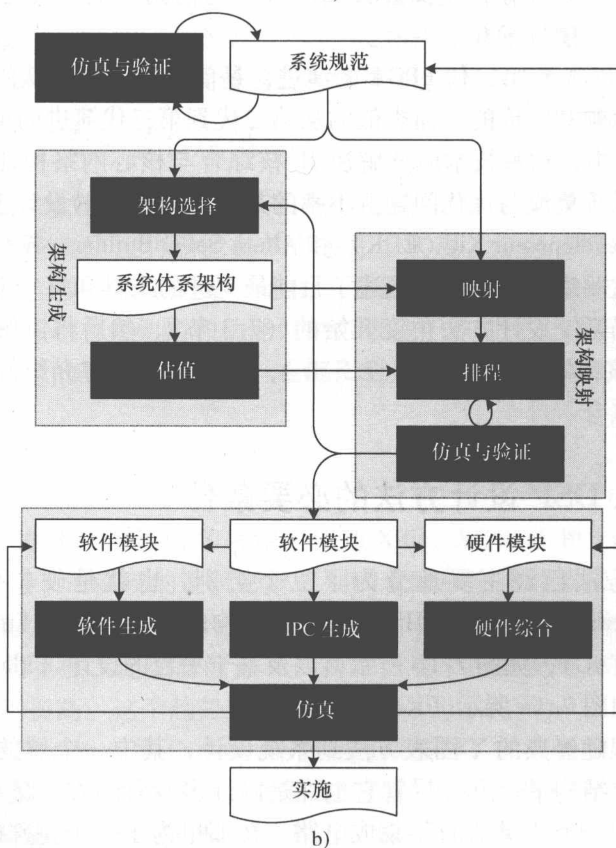
7.3　FPGA DSP设计方法的必要条件

系统设计方法已经主要地分为了三个类别，也就是硬软件的合作设计（Hardware - Software Codesign，HSC）、功能架构的合作设计（Function Architecture Codesign，FAC），如图7-1b所示，以及基于平台的设计（Platform Based Design，PBD），如图7-1a所示（Keutzer等）。

PBD理念跟随经典的Y图表方法到系统设计，其中一个域的算法被映射到一个相对固定的结构平台上（尽管它也许是以许多专有的方法量身定做的）。这个平台被当作是一个“灵活的”集成电路，在那里为了一个特殊的应用客制化，是通过对一个或更多的芯片组件“编程”来实现。



a)



b)

图 7-1　PBD 和 FAC 设计流程

a）PBD 图表　b）FAC

7.4　系统详述

在 7.1 节中应该注意的是在所有系统设计过程中一个关键的共同方面是计算类语言的明确定义的模式中应用的模式的。在接下来设计过程和工具的介绍中，很明显这些都遵从这个原则。

7.4.1　Petri 网

Petri 网（Murata 1989）是一个加重的、定向的二分图表，包含位置和转变，如图 7-2 所示。

随着符号经转换在位置之间移动，位置中（图 7-2 中的 $P_1 \sim P_4$）包含了符号。如果转换的输入位置有一个规定的符号最小数量，那转换（图 7-2 中的 T_1 和 T_2）是可行的，在最小数量点上，符号的规定数量会被移出输入位置，并且一个规定数量会被插入到输出位置中。一个 Petri 网的状态通过位置中符号的配置来定义，与网络的激励或转换规则一起决定来自目前状态的下一个状态。

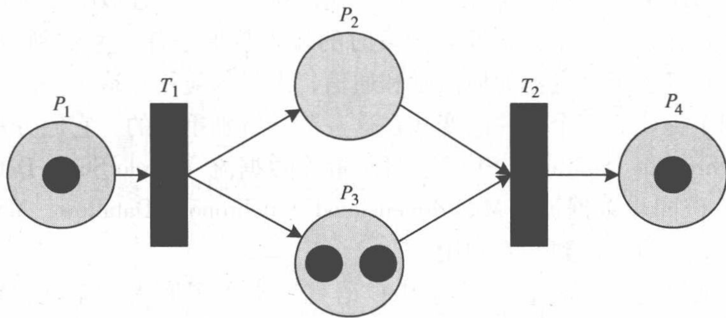


图 7-2　Petri 网络

7.4.2　进程网络和数据流

目前最流行的数据流语言的根坐落在 Kahn 进程网络（Kahn Process Network，KPN）模式（Kahn 1974）中。KPN 模式描述了经单向的 FIFO 队列的一套并行进程，或者“计算站”通信。一个计算站通过使用局部存储器，在一个或所有它的输出线路上产生输出来消耗沿着它的输入线路到来的数据符号。在 DSP 系统中，符号通常是数字化的输入数据值。持续地输入到系统中产生输入数据流，促进了计算站在系统输出上产生数据流。KPN 的一般结构如图 7-3 所示。在 KPN 中每个输入样值具体计算功能重复应用的语义使得这个建模方法很好地匹配了 DSP 系统的运行习惯。

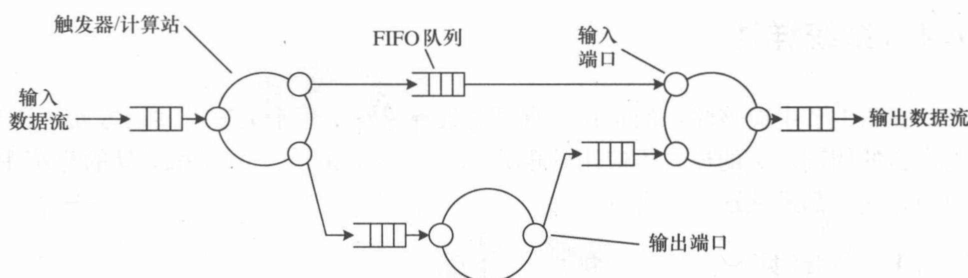


图 7-3 简单的 KPN 结构

在数据流进程网络（Dataflow Process Network，DPN）模式中（Lee 和 Parks 1995），KPN 模式与语义因为计算站（在这里作为一个触发器）的运转一起得到扩张。一系列参与的激励被定义为一个特别的 Kahn 进程的类型叫作数据流进程，在那里每个激励都从输入符号映射到输出符号，并且连续地映射输入流到输出流。制定激励规则是为了确定每个触发器应该怎样激发及什么时候激发。特别的，触发器激励会耗尽输入符号并且产生输出符号。一套连续的激励规则因为每个触发器而存在，并且定义触发器激发时的输入数据条件。考虑到 DPN 坚实的计算基础，它描述了触发器如何激励和通信，大量的应用在这个一般主题上的改进已经得到了提议。三个特殊的变式在这一节是特别重要的，它们分别是同步数据流（Synchronous Dataflow，SDF）、环形静态数据流（Cyclo Static Dataflow，CS-DF）和多方位同步数据流（Multidimensional Synchronous Dataflow，MSDF）它们的功能在后面章节会得到开发利用。

正如先前概述的，大量不同的 MoC 语言通常为了嵌入式系统，特别是 FP-GA，在现代系统级设计工具中得到了普遍利用。确实，比如统一的建模语言或 UML 的规范化成果企图将这些模式的相互作用规范化。UML 拥有各个领域的系统说明，它们每一个都揭示了不同的系统特点并且都适合于描述一个不同类型的系统。虽然有大量具体的系统例子可能比手编程序的效率还低，但在一系列系统中，迅速的实现仍然会产生足够有效率的产品，这节约了设计时间。大量的这种方法对单处理器的软件综合开发出这些技术，并且多处理器的架构在接下来会有概述。

7.4.3 嵌入式多处理器软件综合

许多联系着 UML 的工具，比如来自 Artisan 的 Real Time Studio（Artisan Software Tools Ltd 2004）和来自 I-Logix 的 Rhapsody 能从这些模型中生产嵌入式代码。其他方法使用域的具体图解系统来描述。MATLAB® Real Time Workshop

(RTW) 为直接来自 Simulink®图示系统描述的大量目标类型提供了生成代码的能力。考虑到来自 FPGA 技术的 Simulink® (Hwang 等　2001) 硬件综合的可用性，为了分级系统设计将 MATLAB®的工具集放在一个非常重要的位置上，虽然集成分级环境并不存在。

交替变换方法，比如来自 I - Logix 和 Ptolemy Classic (Madahar 等　2003) 的 Rhapsody 和 Statemate 工具 (Gery 等　2001，Hoffmann 2000)，提供了来自各种具体域的高级系统描述的代码生成能力。当每种方法提供具体的迅速实现能力时，没有一个是完整的系统级分级 FPGA 设计的解决方法。许多工具都提供一种不同类型的方法，在那里，应用在一个 MoC 中得到描述 (为了提供一个平台独立的 DSP 算法的描述)，这是在连接到一个 API 来使算法自动编码的实现能够移植到目标平台之前。这种方法的典型就是 GEDAE。

7.4.4　GEDAE

GEDAE 是一个图示的数据流算法规范及嵌入式的快速多处理器实现环境。GEDAE 发展环境的结构如图 7-4a 所示。系统中的每个处理器通过一个板级支持包 (Board Support Package，BSP) 被赋予特征，它是平台 API。流图编辑器连接了一套在一个 DFG 格式中的基本体 (应用的基础叶子功能) 并且对它们进行编制。基本体如强加语义的结构化的 C 函数一样来表达触发器的数据流本质。

为了高级系统的最优化，许多图形转换得到了应用，正如将在第 11 章描述的。DFG 在平台里是跨越处理器来分区的，并且许多基原是即时插入的，比如处理器互相通信 (发送/接收) 的基原、具有一个调度表中的数据运动的拷贝盒子及具有数据流基原同步的同步基原。最后的结构定于生成程序来在目标处理器上执行。为了高级系统最优化，大量的图形变换得到应用。

程序调度的结果是一个发射包，包含了对实现平台里的微处理器来说可执行文件和每个处理器的调度信息。为了改进过的在嵌入式处理器上的函数性能，调度表能利用一套最优化的向量库函数。为了每个处理器的分布式应用，调度表通过驱动运行时内核的数据来执行。在主机上编制的命令与分布式命令一起传递修改甚至动态重配置的数据，有去向分布式应用的所有元件通道。GEDAE BSP 的一部分在没有程序命令的前提下允许嵌入式处理器间的通信。当应用得到部署后，命令程序会被移除，并且主系统读取发送包，初始化各种平台处理器，然后应用运行分布到各个处理器。

这个目前在为了军事应用而迅速成型的方法中扮演着重要角色。图 7-4b 所示为在 ESPADON 设计流中 GEDAE 的位置，一个主要的欧洲的合作结果首先涉及 BAE 系统和 Thales。

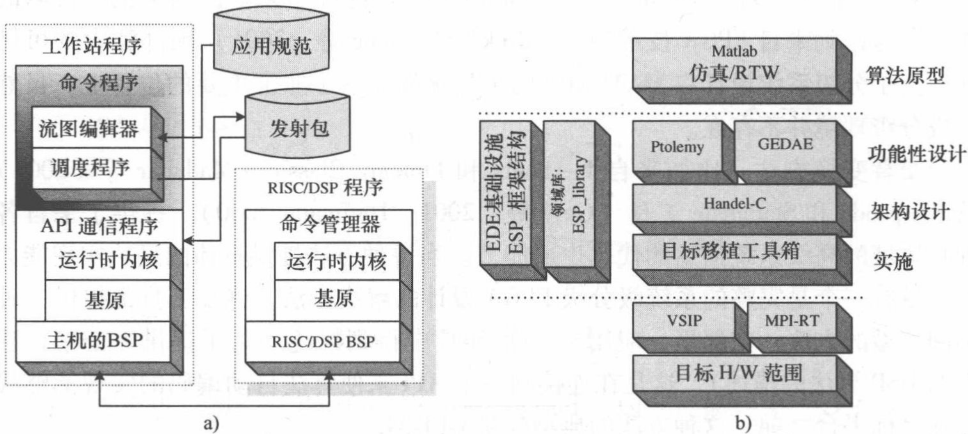


图 7-4　在 ESPADON 系统级设计环境中的 GEDAE
a）GEDAE 发展环境　b）ESPADON 环境

7.5 FPGA 的 IP 核生成工具

7.5.1 图解 IP 核发展途径

基于模块的工具从模块框图生成 VHDL 或 Verilog HDL 代码来支持硬件设计。代码被馈入一个硬件综合工具中来在一个 FPGA 或 ASIC 中实现 DSP 的设计。基于模块的方法仍然要求设计者熟悉核心的定时和控制方面，能够执行 FPGA 设计流的后期处理程序除外。此外，对设计者仅仅可利用的模块是标准 IP 核库。系统设计者必须同样熟悉底层硬件，这都是为了在硬件中有效实施 DSP 算法。

大多数这些工具代表了流行的 Simulink® 和 MATLAB® 上的信号处理算法，而且通过生成 VHDL 或 Verilog HDL 代码，能使来自这些高级描述中的 FPGA 实现成为可能。Simulink®模块库用来表示公共信号处理函数。在使用者使用这些特别的模块来发展他们的算法之后，FPGA 工具能转换设计到 FPGA 的实现。这些工具包括了 Xilinx's System Generator 和 Altera's DSP Bulider。

系统发生器（Xilinx Inc. 2000）使用来自 MathWork Inc. 的流行的 MATLAB® Simulink®工具及通过 Xilinx Inc. 发展的核心来给予一个强大的高级建模环境，它能被用于 DSP 系统设计。这个算法通过使用 Simulink®得到描述，最初通过使用双倍精度算法来实现，然后向下微调到固定的点处，并且转换成硬件实现。系统发生器包括一个叫作 Xilinx Blockset 的 Simulink®库和软件来将一个 Simulink®模型转换成这个模型的硬件实现。此外，系统发生器自动地为 FPGA 综合产生命

令文件、HDL仿真及实现工具，因此允许使用者完全在一个图解的环境中工作。Xilinx Blockset能自由地与其他Simulink®模块结合，但是仅仅Xilinx模块和子系统可以被转换成硬件。系统发生器通过映射Xilinx Blockset的子系统到IP库模型来实现，并且将Simulink®层级转换到一个分级VHDL网络表中。

当允许快速算法级别说明时，比如能对最后的系统性能有主要影响的核心数据通道的潜在因素、流水线级别和数值截断的实现问题没有得到考虑。目前，当使用核心时，就需要使用者来结合这些问题相应地修改模型，这是值得考虑的。此外，Xilinx IP核的标准库对使用者来说仅仅是可利用的模块。其他“黑匣子”核心能通过逻辑设计师使用标准HDL技术来得到发展，但是目前这些不能在同样的环境中被建模。当为了将DSP工具与Xilinx System Generator™结合时，新的AccelDSP 8.1综合工具提供了DSP算法，具有高性能DSP系统，该设计流程使用MATLAB®和仿真器®设计工具实现强大的系统设计能力。

7.5.2 Synplify DSP

Synplify®DSP是一个IP核架构的综合工具，它产生来自低级的IP核，经MATLAB Simulink®算法与仿真环境相连的RTL级核心架构。设计者发展他们在Simulink®中的算法模型并且逐渐地经大量步骤将它改善到一个专用硬件FPGA架构中，如图7-5所示。正如所显示的，起点是理想的算法模型，合并了数字表示系统的任何方式（比如浮点、双精度等）。这会被改善为一个在Synplify DSP设计环境中的定点等值物。每个在Simulink®图标上的触发器都被映射到一个提供工具的Simulink®模块集的触发器上，这代表那个触发器核心的运转情况。

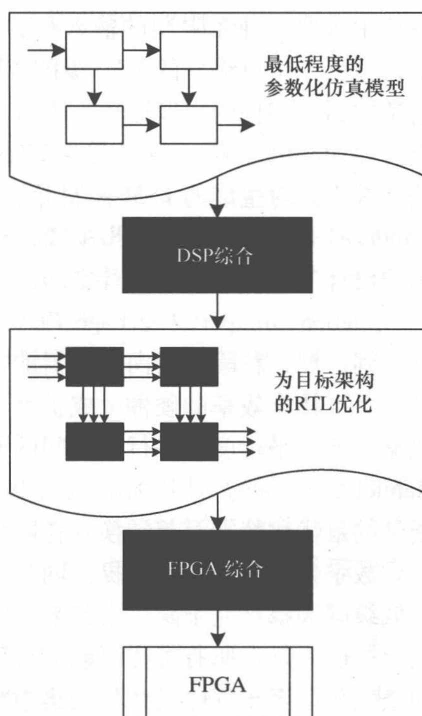


图7-5　Synplify® DSP核心综合

为了对理想算法的运行情况与等值核心的物理运转情况之间的差异负责，许多自动架构的最优化在图7-5中的DSP综合阶段是可行的，包括了流水线法、重新定时及架构

的自动的向量化。最后落定的架构被转换为经传统 RTL 级综合在设备上实现的 RTL 等值物。

进一步的特征，包括了对状态机运行情况进行微调的设计和经来自 MATLAB M 代码的子集规范的综合架构，允许对架构和运行情况的细粒度最优化。

7.5.3 基于 C 语言的迅速 IP 核设计

C 语言通常都因为大量开源代码的存在而被用于描述 DSP 算法，特别是基于标准的应用。此外，由于通过 C 语言提供更高级别的抽象，因此当与一个传统的 Verilog 或 VHDL RTL 设计方法比较时，C 语言的设计获得了重要的生产率。Mentor Graphics Catapult C 和 Celoxica’s Handel-C 是两种主导的工具，通过使用 C 语言，它们使得设计者能够实现目标 FPGA 产品。基于 C 语言的 FPGA 设计工具在所有开发的 FPGA 架构中也有它们的限制。FPGA 构造了比如 LUT、移位寄存器的逻辑和流水线，它们都需要为实现高性能设计而深入掌握。

SPARK 是一个为了应用并行编译转换（Gupta 等 2003）的高级综合架构。SPARK 的影响是高质量的综合结果，是因为有着复杂控制流的设计而产生。在 ANSI - C 中采取一种无限制的输入行为说明，通过解析它来给予一个层次中间的表示，并且产生可综合的 RTL 级的 VHDL。在调度时，一系列转换得到使用，其中包括数据依赖性的提取途径、平行代码移动技术、循环流水线的基础运转及一些支持编译器的途径（Gupt 等 2003）。在调度后，SPARK 系统执行资源分配、控制综合并对生成的 FSM 控制器最优化。

Handel-C 是在硬件中（FPGA 或 ASIC）实现算法的一个高级编程语言，并且伴随的设计工具允许架构设计空间的开发和硬件/软件协同设计（Oxford Hardware Compilation Group 1997，Page 1996，Celoxica Ltd 2002）。C 语言的一个小子集在此得到扩展，移除了面向处理器的特点，比如指针和浮点算法。它为了配置硬件设备及支持有效率的硬件生成扩展了一些构造。这些硬件设计扩展包括了灵活的数据宽度、平行度及并行元件间的通信。

Handel-C 能够通过使用所有的公共表达式说明复杂的算法，并且将它们与一些硬件的最优化特点一起转换到在网络列表级的硬件中。不幸的是，Handel-C 仅仅允许数字化设计，而却不提供同步电路及高度专业化硬件特点的设计。早先说明的低级问题都被完全隐藏了起来，因为它的焦点在快速原型设计和算法级的最优化上，而不是在所有潜在可能的设计特点的研究上。编译器实行所有门级的决定和最优化以至于程序设计员能集中精力在设计任务上。

7.5.4 基于 MATLAB® 的快速 IP 核设计

虽然 C/C ++ 已经成为一种综合语言的流行选择，但使用者也许会因为以下

的因素（Haldar 2001）在一些方法里使用 MATLAB®。第一，MATLAB®提供了更高级别的抽象且比 C 语言的开发时间更少。第二，MATLAB®有一个被良好定义的信号/图像处理函数模块集，它有着一组丰富的与矩阵运算相关的库函数。在 MATLAB®中，最优化更加简单并且由于指针和其他复杂数据结构的缺少编译器也更不易于出错。最后，来自 MATLAB 提出的并行比 C 语言更加简单，因为来自 C 语言循环并行的自动提取要受到复杂的数据相关性分析，然而，在 MATLAB®中，DSP 算法被表达为非常服从并行化的矩阵运算。

AccelFPGA（AccelFPGA 2002）提供了 DSP 算法制造和 FPGA 硬件设计之间缺少的一环。它产生来自 MATLAB®和 Simulink®在目标 FPGA 的内部执行资源、路由架构和物理设计方面最优化且综合的 RTL 模型。这个由设计师设计的产能工具是基于 MATCH 的（分级计算系统（Haldar 2001）的 MATLAB®编译器)。它帮助设计者来为可配置计算系统（Banerjee 等　1999）开发有效率的代码。在 MATCH 项目中（Haldar 2001)，编译器生成硬件和手动设计硬件之间在性能上的比较，显示了手动设计的硬件在性能上有很高的优化。因此，随着 AccelFPGA 为了提高设计入口的抽象级别，更高级别的抽象不能为了效率的实现而处理底层要求的复杂性，并且因此这也是设计时间和质量之间的权衡。

7.5.5　其他快速 IP 核设计

Stnopsys Behavioral Compiler™（Synopsys Inc. 2000）是一种行为的综合工具，它允许设计者评估供选择的架构，迅速生成一个最佳的门级实现，并且制造一个由数据通路、存储 I/O 和控制 FSM 组成的设计。Behavioral Compiler 包括了链接、多重循环操作、流水线操作、环路流水线、动作的重定时等。链接调度多重的内部迭代的相关运算到一个简单循环中去，在那里个体运算符的总延时比给出的时钟周期更少。多重循环运算花费多于一个简单循环，并且自动调度是超过要求的周期数的。因此 Behavioral Compiler 能从一个单一规格创造出多重架构，并且通过使用高级约束条件权衡生产量和延时。

在 C 语言和 Mathematica 中的 MMAlpha 是一个操纵和转变 ALPHA 编程语言的编程环境（Derrien 和 Risset 2000)。它提供一个从算法的高级功能性说明到一个可综合的 VHDL 编程的通道。MMAlpha 把 ASIC 和 FPGA 作为目标，并且是一个功能性的数据并行语言，它允许递归方程和脉动阵列的硬件架构的表达式（Dupont de Dinechin 等　1999)。ALPHA 程序先进的操作包括流水线技术、基础的改变、标准化和排程。流水线技术是一个广泛使用在脉动综合中的转换。它通过改变 ALPHA 中的基础来实现。ALPHA 也为重新定时同步架构提供了语法。为了产生一个 ALPHA 规格的硬件，ALPHA 的子集 ALPHARD 得到了使用。ALPHARD 使一个有规律架构的结构定义成为了可能，比如给出的脉动阵列和在

RTL 级生成的 VHDL 代码。但是，由 ALPHARD 程序表示的，仅仅只有 1D 并不是 2D 脉动阵列能被转换成 VHDL 代码。

JHDL（Bellows 和 Hutching 1998）是一套基于 Java HDL 的 FPGA CAD 工具，这最初是为了可重构系统，JHDL 在 Brigham Young 大学得到了发展。JHDL 的主要目标是为了开发一个说明电路的工具，它能随着时间动态地被改变。它允许使用者开发高性能的可重构计算的应用。JHDL 是基于 Java 的，并且是一个结构的设计环境，结果就是形成了与那些使用传统行为得到发展的综合工具相比，更小更快的电路。它支持设备独立性的某一等级，并且使用 TechMapper 来针对特别的 FPGA 平台。JHDL 假设了一个全局同步的设计，而且不支持多重时钟同步和异步环路。此外，JHDL 不支持行为的综合。与 Handel - C 相比，它是一个更低的级别并且提供更多在硬件中对实际实现的控制，但是又是一个更加困难的设计环境。

7.6 FPGA 的系统级设计工具

7.6.1 Compaan

这项研究成果包括三个主要工具，即 Compaan，LAURA 和 ESPAM。Compaan /LAURA（Stefanov 等 2004）是一个系统级的设计及最优化方法，它遵从了 PBD 的设计准则。Compaan 是为了从由传统的连续语言编写的嵌套循环的表示中产生 DSP 应用的基于流的功能（Steam Based Function，SBF）模型（图 7-6b 所示为有着结构节点的各种进程网络（Kienhuis 和 Deprettere 2001））的一个自动的工具集，比如 MATLAB®或 C++，如图 7-6a 所示。SBF 算法的表示能如在 FPGA 上的专用硬件网络一样经一个范例得到实现，它生产设计前 IP 核的封装器，如在 Harriss 等（2002）中描述的一样。近年来，这项成果已经扩展到生产实现最优化的 LAURA 工具，以及经一个交叉开关组件为多重处理器通信代码生成的 ESPAM 工具。系统最优化主要通过开发连续的代码并行技术来实行，比如循环展开或者循环偏移，如在 Stefanov 等（2002）中概述的。

7.6.2 ESPAM

ESPAM 系统综合工具将由 Compaan 提议的设计进程扩展到了针对分级多重处理器平台。此外，设计进程是从一个在 MATLAB 或 C++或者一个等值编程语言中规范的连续应用开始的，这个应用自动地通过 KPNGen 工具（Verdoolgege 等 2007）被转换到一个并行 KPN 规范中，这个规范已经增加了处理所谓弱动态程序类型和评估 KPN FIFO 缓存区大小的能力。最后的 KPN 映射到一个多重处

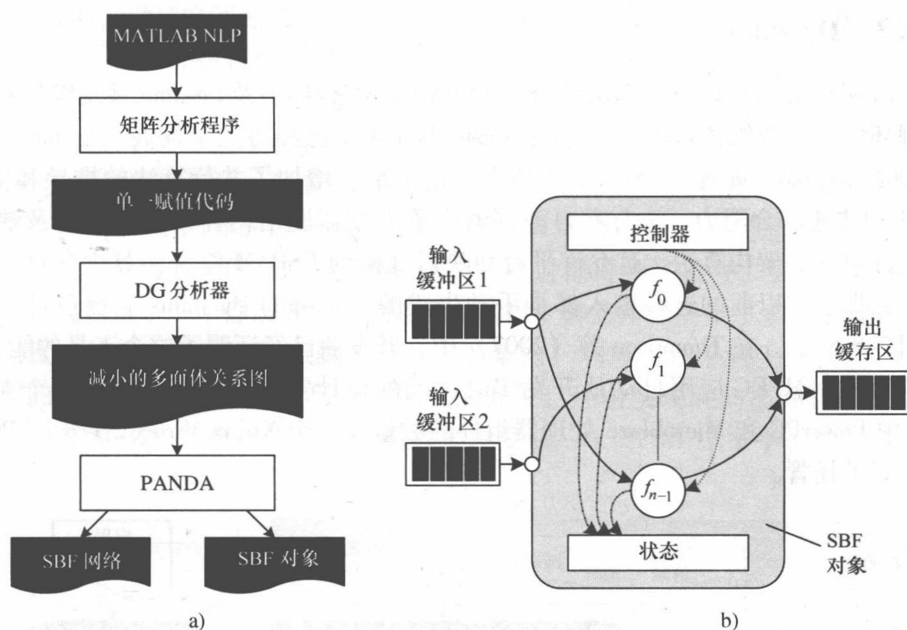


图 7-6 Compaan SBF 网络推导

a）比较 SBF 的推导 b）SBF 节点结构

理器片上系统（Multiprocessor System on Chip，MPSoC）平台规范中。平台上的每个处理器的源连同一个交叉开关、基于总线或点对点处理器间的通信架构（Nikolov 等 2006）一起生成。系统最优化经转换输入连续的规范到一个 KPN 进程中的各种优化转换的应用得到实现。图 7-7 所示为 ESPAM 设计方法的综述。

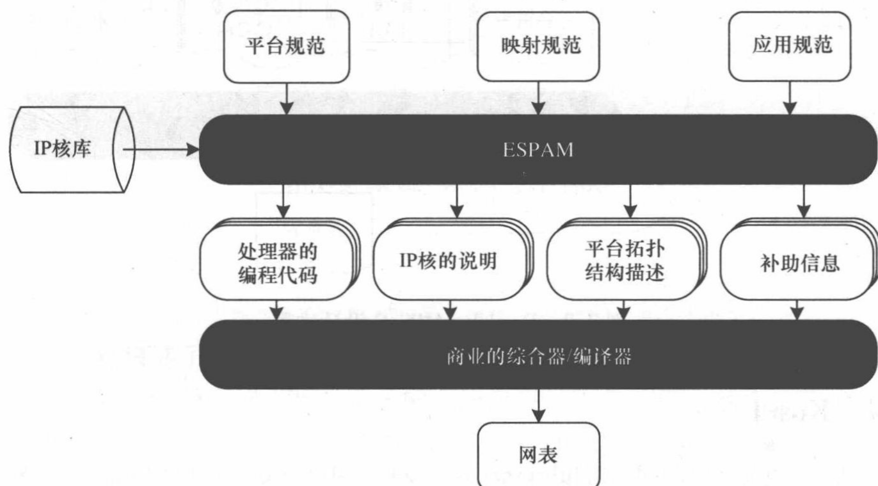


图 7-7 ESPAM MPSoC 设计进程

7.6.3 Daedalus

Daedalus MPSoC 设计方法结合了 ESPAM 的综合能力及 Sesame 系统级仿真和探测环境的系统级仿真能力，这在 Amsterdam 大学已经得到了发展（Pimentel 等 2006，Thompson 等 2007）。本质上，这个结合增加了并行算法的推导和 ESPAM 的快速综合能力，也有着自动探索多重处理器网络拓扑结构的能力及映射到设计空间的架构算法。这允许平台和映射规格到 ESPAM 综合工具中的自动生成，在那里，以前的这些输入都是手动生成的。全部的 Daedalus 系统设计框架如图 7-8 所示。在 Thompson 等（2007）中，开发商已经证明了这个工具的能力，为了为一个 MPEG 应用自动地开发实时性能的设计空间，通过改变在一个 MPSoC 中 PowerPC 和 Microblaze 处理器组件的数量在一个 Xilinx Virtex 2 Pro 2 FPGA 上得到了托管。

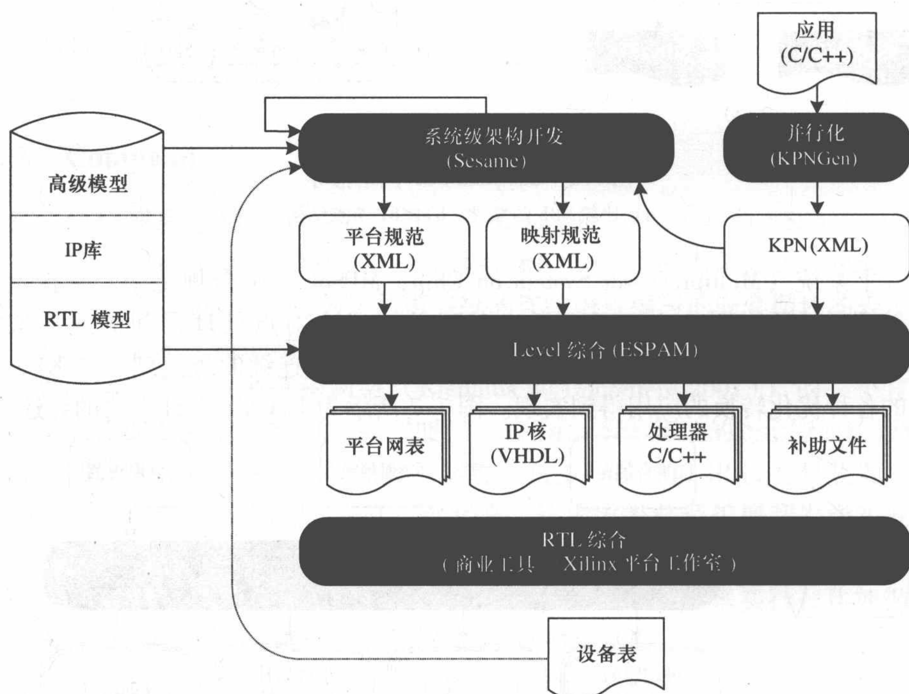


图 7-8 Daedalus MPSoC 设计进程

7.6.4 Koski

Koski 多重处理器 SoC（Multiprocessor SoC，MPSoC）设计环境是一个统一基

于UML的为设计空间开发及MPSoC的自动综合、编程和原型设计的基础设施，其首要针对的是无线传感器网络的应用。Koski环境结构如图7-9所示。

Koski有一个基于组件的设计准则，与最后的架构是一起的，这个架构包含了大量的嵌入式处理器和到一个HIBI总线网络的IP核接口（Salminen等2006）。正如所显示的，到Koski的输入是应用行为、架构和设计约束条件的手工规范。Koski中的算法规范是在7.4.2节KPN建模语言中论述过的。由此，算法和架构的模型及KPN触发器到平台上的映射都得到了推导。如图7-9所示，架构开发包含了一个静态和一个动态两个阶段。静态阶段决定了多重处理器资源的配置（从单个到多重处理器各种可能的架构）和应用在平台上的映射，以及实现的调度。在动态阶段，启用了细粒的开发，经由实现时间准确建模。Koski提供软件和RTL综合技术来使最后的实现变成一个FPGA原型平台的样板成为可能。

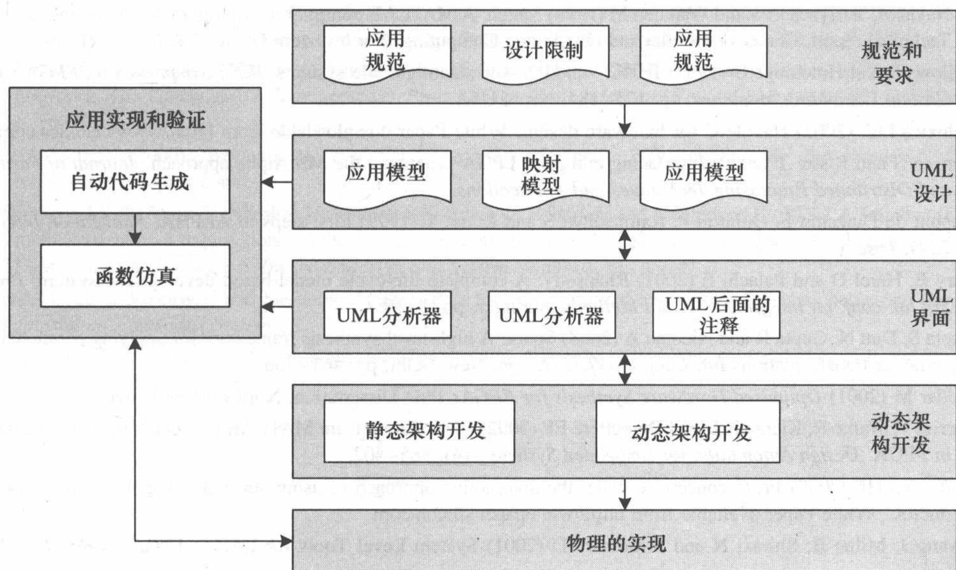


图7-9 Koski MPSoC设计环境

7.7 总结

本章已经概述了在应用建模、设计方法论、IP核和核心网络综合、多重处理器软件综合，以及分级FPGA系统的系统级设计工具领域中的尖端技术。其表明了所有这些方法的快速实现能力都起源于在应用的最典型的MoC中良好地定义了语义的开发，以此来生产嵌入式硬件和软件的实现。

当考虑到在不同领域中的快速实现工具时，值得注意的是它们有共同的单个的全局相似点。第 8 章将考察为高度流水线的 IP 核自动地基于组件的综合，开发信号流建模的技术。它启示了在这种情况下，比较固有数据流建模语义的处理与在比如 GEDAE 的多重处理器软件综合工具中的处理。第 11 章将概述它们有多么的不同，并且不需要补充的方法，虽然这两种方法有同样的出发点。此外，本章还概述了为系统共同的综合设计共同出发点的技术，这是比通过使用这些目前不同的技术来实现更加有效的。

参考文献

AccelFPGA (2002) AccelFPGA: High-level synthesis for DSP design. Web publication downloadable from http://www.accelchip.com/.

Artisan Software Tools Ltd. (2004) Artisan real-time studio. Datasheet, available from http://www.artisansw.com/pdflibrary/Rts S.O datasheet.pdf.

Banerjee P, Shenoy N, Choudhary A, Hauck S, Bachmann C, Chang M, Haldar M, Joisha P, Jones A, Kanhare A, Nayak A, Periyacheri S and Walkden M (1999) Match: A MATLAB compiler for configurable computing system. Technical report, Center of Parallel and Distributed Computing, Northwestern University. CPDC-TR-9908-013.

Bellows P and Hutchings B (1998) JHDL – an HDL for reconfigurable systems. *IEEE Symposium on FPGA's for Custom Computing Machines*, pp. 175–184, Napa, USA.

Celoxica Ltd. (2002) Handle-C for hardware design. White Paper downloadable from http://www.celoxica.com.

Derrien S and Risset T (2000) Interfacing compiled FPGA programs: the MMAlpha approach. *Journal of Paralle and Distributed Processing Techniques and Applications*.

Dupont de Dinechin F, Quinton P, Rajopadhye S and Risset T (1999) First steps in ALPHA. *Publication Interne 1244, Irisa*.

Gery E, Harel D and Palachi E (2001) Rhapsody: A complete life-cycle model-based development system. *Proc. 3rd lat. conf. on Integrated Formal Methods*, Springer, pp. 1–10.

Gupta S, Dutt N, Gupta R and Nicolau A (2003) Spark: A high-level synthesis framework for applying parallelizing compiler transformations *Int. Conf. on VLSI Design*, New Delhi, pp. 461–466.

Haldar M (2001) *Optimised Hardware Synthesis for FPGAs* PhD Dissertation, Northwestern University.

Harriss T, Walke R, Kienhuis B and Deprettere EF (2002) Compilation from MATLAB to process networks realised in FPGA. *Design Automation for Embedded Systems* 7(4), 385–403.

Hoffmann HP (2000) From concept to code: the automotive approach to using statemate magnum and rhapsody microC. White Paper available from http://whitepaper.silicon.com.

Hwang J, Milne B, Shirazi N and Stroomer JD (2001) System Level Tools for DSP in FPGAs. *Proc. 11th Int. Conf. on Field Programmable Logic and Applications*, pp. 534–5438.

IRTS (1999) *International Technology Roadmap for Semiconductors*, 1999 end. Semiconductor Industry Association. http://public.itrs.net

Kahn G (1974) The Semantics of a Simple Language for Parallel Programming *Proc. IFIP Congress*, pp. 471–475.

Kangas T, Kukkala P, Orsila H, E. S, Hannikainen B, Hamalainen T, Riihmaki J and Kuusilinna K (2006) Uml-based multiprocessor soc design framework. *ACM Transactions on Embedded Computing Systems (TECS)* 5, 281–320.

Keutzer K, Malik S, Richard Newton S, Rabaey JM and Sangiovanni-Vincentelli A (2000) System level design: Orthogonolization of concerns and platform-based design. *IEEE Trans. CAD* 19, 1523–1543.

Kienhuis B and Deprettere EF (2001) Modelling stream-based applications using the sbf model of computation *Proc. IEEE Workshop on Signal Processing Systems (SIPS)*, pp. 291–300.

Lee EA and Parks TM (1995) Dataflow Process Networks. *Proc. IEEE* 85(3), 773–799.

Lee EA, Neuendorffer S and Wirthlin WJ (2003) Actor-oriented design of embedded hardware and software systems. *Journal of Circuits, Systems and Computers* 12, 231–260.

Madahar BK, Alston ID, Aulagnier D, Schurer H and Saget B (2003) How rapid is rapid prototyping? Analysis of ESPADON programme results. *EURASIP Journal on Applied Signal Processing* 2003(1), 580–593.

Marsh P (2003) Models of control. *IEE Elect. Syst. and Software* 1(6), 16–19.

Murata T (1989) Petri nets: properties, analysis and applications. *Proc. IEEE* 77(4), 541–580.

Nikolov H, Stefanov S and Deprettere E (2006) Multi-processor system design with ESPAM. *Proc. 4th Int. Conf. on Hardware/Software Codesign and System Synthesis*, pp. 211–216.

Oxford Hardware Compilation Group (1997) The Handel language. Technical report, Oxford University.

Page I (1996) Constructing hardware-software systems from a single description. *Journal of VLSI Signal Processing* 12(1), 87–107.

Pimentel A, Erbas C and Polstra S (2006) A systematic approach to exploring embedded system architectures at multiple abstraction levels. *IEEE Transaction on Computers* 55(2), 1–14.

Rowson JA and Sangiovanni-Vincentelli A (1997) Interface-based design *Proc. 34th Design Automation Conference*, pp. 178–183.

Salminen E, Kangas T, Hamalainen T, Riihimaki J, Lahtinen V and Kuusilinna K (2006) Hibi communication network for system-on-chip. *Proc. IEEE* 43(2-3), 185–205.

Stefanov T, Kienhuis B and Deprettere E (2002) Algorithmic transformation techniques for efficient exploration of alternative application instances *Proc. 10th Int. Symp. on Hardware/Software Codesign*, pp. 7–12.

Stefanov T, Zissulescu C, Turjan A, Kienhuis B and Deprettere E (2004) System design using Kahn process networks: The Compaan/LAURA approach *Proc. Design Automation and Test in Europe (DATE) Conference*, p. 10340.

Synopsys Inc. (2000) Behavioural compiler™ quick reference. Web publication downloadable from http://www.synopsys.com/.

Thompson M, Stefanov T, Nikolov H, Pimentel A, Erbas C, Polstra E and Deprettere E (2007) A framework for rapid system-level exploration, synthesis and programming of multimedia MPSoCs *Proc. ACM/IEEE/IFIP Int. Conference on Hardware-Software Codesign and System Synthesis*, pp. 9–14.

Verdoolaege S, Nikolov H and Stefanov T (2007) Pn:a tool for improved derivation of process networks. *Eurasip Journal on Embedded Systems*. 1.

Xilinx Inc. (2000) Xilinx system generator v2.1 for simulink reference guide. Web publication downloadable from http://www.xilinx.com.

第 8 章　基于 FPGA 的 DSP 系统的架构由来

8.1　引言

回顾以往的技术，第 4 章和第 5 章清晰地演示了不管是 ASIC 还是 FPGA 技术平台，在硅硬件上实现 DSP 算法时研发一个电路架构的必要性。这种电路架构能够满足应用的性能要求。正如之前强调的，在 FIR 滤波器表达式中实现高级的可行的并行性是可能的（第 2 章中的式（2-11）），这是为了实现性能的提高，或者使得 SFG 或数据流图（Dataflow Graph，DFG）流水线化。第一种实现是假定硬件资源在硅面积方面是不受限制的，而第二种方法是假设由流水线导致的（在一个更小的时钟周期上是公认的）时钟周期方面增加的延时是能接受的。很明显，在硬件层的最优化对最终的设计有直接的影响。这两方面都能在电路架构中得到满足。

正如第 5 章所描述的，这个取舍使得在比如微处理器、DSP 处理器，甚至可重构处理器这样的“固定架构”的平台上开发更加容易，因为足够合适的工具能够或已经得到研发，从而将算法要求有效地映射到可用硬件上。正如之前所探讨的，使用 FPGA 的主要吸引力是能够改进高级的可用硬件来满足算法的具体需求。但是，这实际上阻止了高效的编译工具的使用，架构的“门柱”已经移动了，因为架构是按需设计出来的。这个事实在第 8 章将得到突出，并且第 8 章也包括了一些商业上或大学和研究所研发的高级工具。因此，对一系列架构的设计方案和在高级抽象下计算出的成本因素一同研究是很典型的一种做法。

本章我们将研究简单 DSP 系统的直接映射，或者更确切来说是比如 FIR 滤波器或 IIR 滤波器、自适应滤波器等的 DSP 组件，因为这些组件现在会成为更加复杂的系统的一部分，比如波束形成器、回波消除器等。关键点是探究应用于 SFG 表示式上的变化如何能影响这种功能的 FPGA 实现，这使得读者能在 SFG 中快速操作，而不是在电路架构中操作。这个趋势经由本书变得更加流行，因为我们试图转移到一个更加高级的表示式上。后面的章节将通过思考系统层面的影响，演示如何使用更加高级别的抽象实现额外的性能提高。

8.2 节将着眼于 DSP 特征并给予一些如何将这些映射到 FPGA 实现上的指导。然后我们讲述 SFG 的改变是如何集中到 Xilinx Virtex FPGA 系列的 FPGA 技术中的。考虑到 FPGA 架构的一个关键方面是分布式存储器，强调了有效率的流

水线技术是一个关键的最优化。这样以详细且正式的方法进行研究，以此来实现要求的流水线等级。本章主要依靠由 Keshab K. Parhi（Parhi 1999）写作的一篇优秀文章，这篇文章包括了 VLSI 硬件中复杂 DSP 系统的实现。然后本章继续囊括大量电路层的最优化，试图使用最小的面积达到必要的速率。这涉及对原始的 SFG 进行变换来达到必要速度，这包括了通过硬件的复制来提高计算量、一个叫作去折叠的过程或在速度允许下共享可用的硬件。在本章中，这些技术被用于简单的例子中，通常是 FIR 滤波器和 IIR 滤波器。

8.2 DSP 算法特点

通过它们特有的本质，DSP 算法趋向于运用在需要处理大量信息的应用中。正如第 2 章所强调的，抽样速率的范围能从语音环境中的 kHz 达到在图像处理应用中的 MHz。清楚地定义有关 DSP 系统的参数是很重要的。

（1）抽样速率能被定义为一种速率，在这种速率下，我们需要处理研究中的系统或算法的 DSP 信号样值。比如，在语音应用中，语音的最大带宽通常被判定为 4kHz，导致抽样速率为 8kHz。

（2）吞吐率定义为一种速率，在这个速率下，数据样值得到处理。在一些场合，DSP 系统的目的是为了匹配吞吐量和采样速率，但是在更低的采样速率系统中（语音和声频），这会导致处理器硬件的利用率不足。比如语音采样速率是 8kHz，但是大量 DSP 处理器的速率都达到几百 MHz。在这些情况下，每秒就需要执行大量计算，那意味着吞吐量 p 要比采样速率高好几个数量级。在吞吐量很高且计算的需要很适中的情况下，就存在重复使用硬件的可能性，也就是 p 次。

（3）时钟速率定义为系统的运行速率。我们认为在一些应用中这个数值可能是错误的，因为内存大小、结构和用法在性能的决定中可能更加关键。在 DSP 系统中，简单地细读一下 DSP 和 FPGA 数据表就可以看出最新 Xilinx Virtex 5 FPGA系列的时钟速率是 550MHz，然而 TI’s C64xx DSP 系列能达到 1GHz。看上去 DSP 处理器比 FPGA 更快，但那是在单个循环中执行计算的数量。它是决定吞吐量的一个主要因素，它能对性能更加准确地评估，但是，还是得取决于实际应用。

因此很明显，我们基本上需要根据吞吐量及采样速率设计系统。这在很大程度上依赖于我们怎么才能有效率地开发电路架构。正如第 4 章和第 5 章中所讲述的，这还是得依靠通过有效管理底层的硬件资源来满足性能要求。对 ASIC 应用，这是一个完全公开的硬件平台，但是对 FPGA，在处理元件方面，例如专用的 DSP 模块、可扩展的加法器结构、LUT 资源、存储器资源（分布式 RAM、LUT RAM、寄存器）和互连性，比如高速 Rocket I/O、可编程互连的各种形式等

还是受限的。其目的是为了将这些资源匹配到计算的需要中，这些最初是基于性能来做的，并且如果吞吐量溢出的话，则权衡一下面积上的取舍。

这的确是现在主要的挑战。在 DSP 处理器中，架构不变的本质就是逐步提高有效率的 DSP 编译器性能来使得如 C 语言的高级语言能够被编译、装配并实现到处理器上。因此 DSP 编译器的目的是为了研究处理模块速率是否允许以要求的采样速率来计算算法的一次迭代。这通过检查处理器固定的资源及调整计算来实现，并以这种方式达到要求的采样速率。实际上，这涉及可用硬件的重复利用，但是我们打算不在这些方面考虑处理器。从 FPGA 的实现中，使用一个即时设计的原因是为了考虑硬件能够重复使用多少次，并且是否能达到这个采样速率？其重点是制造硬件资源来匹配性能要求，这也是本章的一个关键点。

8.2.1 算法特点的进一步描述

采样的一些基本定义、吞吐量和时钟速率现在已经因为 DSP 系统得到了开发。但是，虽然我们开始开发不同技术来改变和提高电路架构的描述方式，但是开发其他定时方面也很重要。比如，使用并发性，以并行和流水线的形式对性能和最终 FPGA 实现的定时产生额外影响。

时延是对应于输入 $x(n)$ 的输出 $y(n)$ 的时间。乍一看，这似乎相当于吞吐量，但是如图 8-1 所示的 $y(n) = ax(n)$ 的计算清楚地演示了事实并非如此，特别是当流水线得以应用时。在图 8-1 中，电路会有三重流水线并且因此在三个时钟周期后将产生第一个输出，因此称作时延；之后，每个周期都将产生一次输出，这就是吞吐量。考虑如图 8-2 所示的简单递归 $y(n) = ay(n-1)$。呈现出来的输出 $y(n)$ 依靠之前的输出 $y(n-1)$，并且因此，时延决定了吞吐量。这意味着如果要花费三个时钟周期来产生第一个输出，则必须等待三个时钟周期来产生每个输出，并且就此而言，这牵扯到了每一个输入。因此很清楚，当为不同的算法推导电路架构时，任何的像流水线能同时改变吞吐量和时延的技术必须小心地考虑。

有许多的最优化能够在 FPGA 实现中得以实现来执行要求的计算，正如以下所列。可以说并行在算法描述中的使用是很正常的，并且不是一个最优化，但在这里主要的定义集中在 FPGA 实现的开发上，一个串行处理器的实现未必会开发这一层的并行。

并行能自然地存在于算法的描述中，也能通过组织计算得到引进，来使得并行得以实现。在图 8-3 中，我们能够实现过程 P_1，P_2 和 P_3，在所有 3 种情况中，如同 3 个独立的处理器 PE_1，PE_2 和 PE_3。在图 8-3a 中，这个过程是来自单个源的，在图 8-3b 中，它们来自分离的源，在图 8-3c 中，过程被连续地组织起来。后者的处理是效率低下的，因为在任何一个时间中仅仅只有一个处理器在运

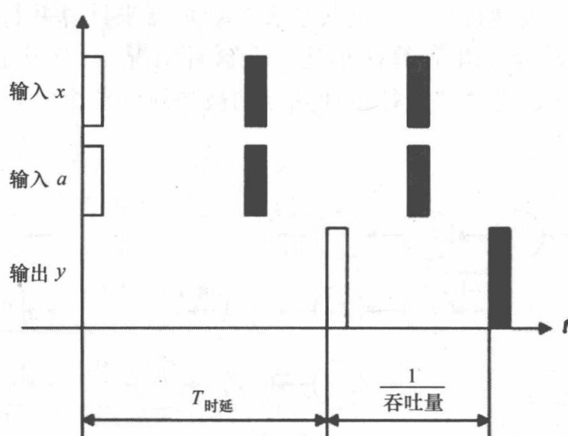


图 8-1　系统 $y(n)=ax(n)$ 的时延和吞吐量的关系

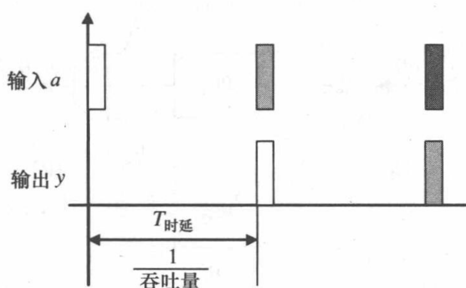


图 8-2　系统 $y(n)=ay(n-1)$ 的时延和吞吐量的关系

转，但是为了表述完整，将其列出。

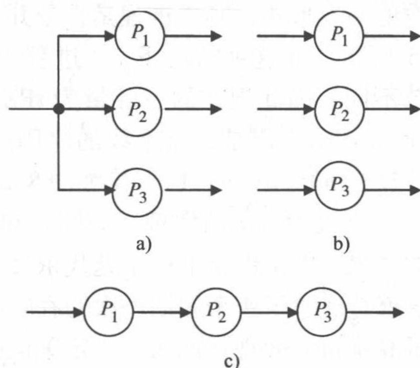


图 8-3　使用三个处理器 PE_1，PE_2 和 PE_3 的算法的实现

a）单源　b）多源　c）顺序算法

交叉能被用于加速计算，通过共享多个处理器来计算并行算法的迭代次数，图 8-4 所示为图 8-3c 的串行算法描述。在这种情况下，3 个进程 P_1，P_2 和 P_3 并行执行算法的 3 次迭代并且每行计算分别被映射到单个的处理器 PE_1，PE_2 和 PE_3 上。

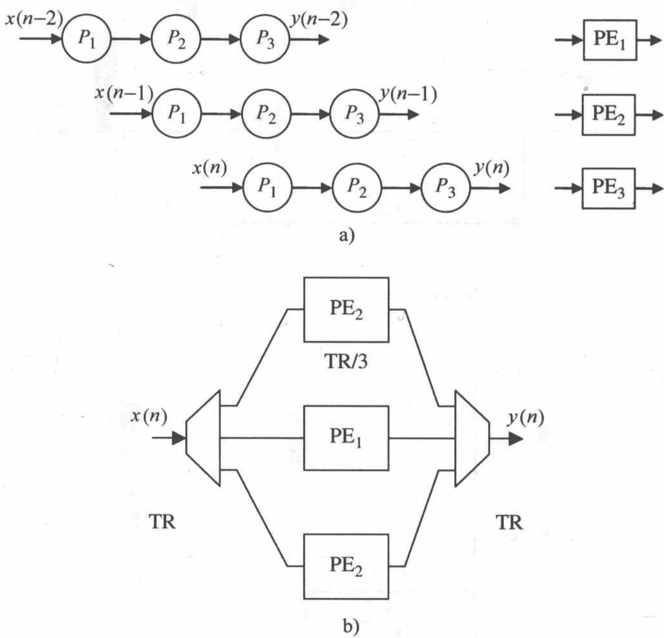


图 8-4 流水线示例

a）交叉演示 b）交叉实现

流水线是另一种有效的并发形式，其中进程是在分开的数据片上执行的，但却是同时执行，如图 8-5 所示。在这种情况下，3 进程 P_1，P_2 和 P_3 都同时执行，但是算法的迭代次数不同。因此现在的吞吐量为 tPE_1，tPE_2 或 tPE_3，而不是图 8-3c 中的 $tPE_1 + tPE_2 + tPE_3$。但是，流水线的应用因为一些递归函数被限制，比如图 8-6 所示的计算 $y(n) = ay(n-1)$。正如图 8-6a 所示，原始的处理器会得到一个时钟速率为 f_c 及吞吐率为 f 的实现。如图 8-6b 所示的四重流水线的应用造成了快 4 倍计时的结果，但是由于下一个迭代依靠当时的输出，因此它将必须等 4 个时钟周期。这就造成了每 4 个周期一次的吞吐率，表明了性能的 0 增长。确实，触发器的设置和维持时间现在成为了决定关键路径的一大要素，而且因此性能也按实际衰减。

很显然，当这些最优化存在时，对一项技术而言不会是一个简单的应用。比如，在最后的 FPGA 实现中使用并行处理及在每个处理器中使用流水线也许是可

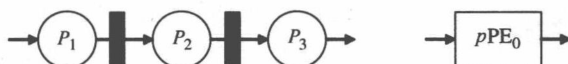
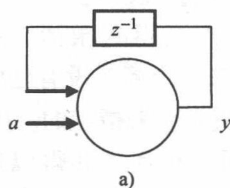
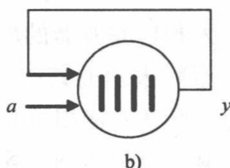


图8-5　流水线的例子

能的。在图8-6b中，流水线不会产生速度的提升，但是现在算法的4重迭代可以交叉，因此实现了一个4倍的提升。很明显，对设计者有许多可用的选择来以最小的面积达到要求的吞吐量，比如串行对并行、并行/流水线和硬件共享的有效利用之间的取舍。本章的焦点是演示设计者如何能在一个算法的表达式中研究这些取舍，从SFG或DFG的描述开始，然后以提高性能为目的执行操作。



a)



b)

图8-6　递归计算 $y(n)=ay(n-1)$ 的流水线

a）原始的递归计算（时钟速率 f_c 和吞吐率 f）

b）流水线版本（时钟速率 $4f_c$ 和吞吐率 f）

8.3　DSP算法的表示

有许多表示DSP算法的方法，从数字描述到方框图，再到HDL描述。在这一节，我们集中研究SFG和DFG表达式，因为我们将这些表达式作为研究一些以上简短概述过的最优化的一个出发点。由于这个原因，提供更多的SFG和DFG表达式的细节是很重要的。

8.3.1　SFG的描述

DSP系统的经典描述是通过使用一个SFG得以实现的。SFG表达式是节点和有向箭头的一个集合，在这个集合中，一个有向箭头(j, k)表示一个从节点j上的信号到节点k上的信号的线性传输。这个箭头通常连接乘法器、加法器或延时元件。表达式为$y(n)=ay(n-1)+x(n)$的经典SFG如图8-7a所示，而方框图如图8-7b所示。DFG表达式如图8-7c所示，并且它在这里被给出是因为它对后面章节的重定时最优化的应用是一个更加有用的表达式。

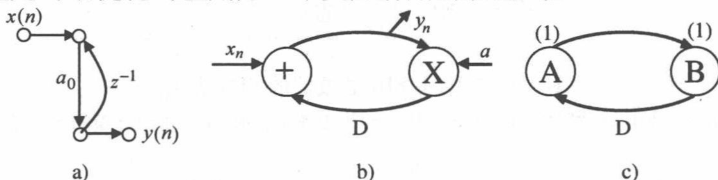


图8-7　简单DSP递归 $y(n)=ay(n-1)+x(n)$ 的不同表示式

a）SFG　b）方框图　c）DFG

8.3.2 DFG 的描述

在 DFG 中，节点表示计算或函数，有向箭头表示与它们有联系的非负数的数据通道。数据流满足了 DSP 算法的数据导向性能，其中，当所有输入数据为可用时，节点会得到激活（执行它的计算）。这产生了优先级的约束（Parhi 1999）。如果一个箭头没有延时，则限制内部的迭代，换句话说激活的顺序是通过 DFG 箭头方向来指示的。如果箭头有一个或更多的延迟，则内部的迭代限制能得到应用，并且当其得以运用时，这个延迟将被转换成一个数字延迟或寄存器。

对 3 抽头 FIR 滤波器的配置是一个很切实的实现。SFG 表示如图 8-8 所示，能被应用于 SFG 表示的其中一种转换是移项。这通过反转所有箭头的方向得以实施，这会交换输入输出节点，同时保持如图 8-8b 所示箭头的增益或箭头的延迟不变。重组的类型如图 8-8c 所示。主要的不同是 $x(n)$ 输入的数据流在没有造成任何对最终 SFG 功能性改变的前提下已经被反转。后面将会介绍图 8-8c 的 SFG 对于应用流水线来说怎样才是一个更加合适的结构。

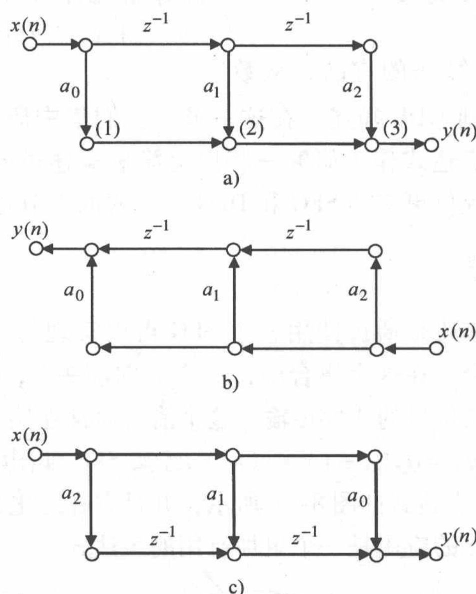


图 8-8 3 抽头 FIR 滤波器的 SFG 表示

a）3 抽头 FIR 滤波器的 SFG 表示 b）移项的 SFG 表示 c）重组移项的 SFG 表示

图 8-8b 所示的 SFG 的数据流表示如图 8-9 所示。在图 8-9 中，被标上 a_0，a_1 和 a_2 的乘法器表示有着两重流水线阶段的流水线乘法器。被标为 A_0 和 A_1 加

法器代表了有着一重流水线阶段的流水线加法器。D标签表示大小等于字长的单个寄存器（在DFG表示式上没有指出）。以这种方法，数据流描述对一个硬件的实现有很好的指引，很明显那是满足研发需要的性能要求合适的SFG表示的一个主要议题。至于流水线架构，为了实现性能要求，这是应用合适的重定时方法来开发流水线正确等级的一个主要的例子。下一节将完全致力于重定时，因为将要出现递归架构，也就是那些涉及反馈回路的结构会出现特别的问题。

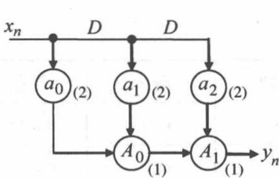


图8-9 简单DFG

8.4 FPGA上映射DSP系统的基础

实现FPGA的一个主要目的是为了决定需要的流水线的重数。标记为（1）、（2）和（3）的节点通过图8-8a的3抽头FIR滤波器的数据时序见表8-1。我们能如图8-8a中添加一个延迟到每个乘法器输出上，这如表8-2一样改变了数据调度。注意到时延现在得到了增加，但结果就是一个周期内无法使用。但是，添加另一个延迟到加法器的输出上会造成如表8-3指出的故障。这是因为一个进程，通过这个进程，我们添加的这些延迟必须通过重定时技术的应用以系统的方式实现。很明显，重定时被正确地应用在第一个例子中，但由于它没有改变电路的功能性，因此在第二个例子中就变得不正确了。重定时经如Kung（1988）中描述的切割原理得以应用。

表8-1 FIR滤波器时序

时钟	输入	节点1	节点2	节点3	输出
0	$x(0)$	$a_0x(0)$	$a_0x(0)$	$a_0x(1)$	$y(0)$
1	$x(1)$	$a_0x(1)$	$a_0x(1)+a_1x(0)$	$a_0x(1)+a_1x(0)$	$y(1)$
2	$x(2)$	$a_0x(2)$	$a_0x(2)+a_1x(1)$	$a_0x(2)+a_1x(1)+a_2x(0)$	$y(2)$
3	$x(3)$	$a_0x(3)$	$a_0x(3)+a_1x(2)$	$a_0x(3)+a_1x(2)+a_2x(1)$	$y(3)$
4	$x(4)$	$a_0x(4)$	$a_0x(4)+a_1x(3)$	$a_0x(4)+a_1x(3)+a_2x(2)$	$y(4)$

表8-2 改进的FIR滤波器时序

时钟	输入	节点1	节点2	节点3	输出
0	$x(0)$	$a_0x(0)$			
1	$x(1)$	$a_0x(1)$	$a_0x(0)$	$a_0x(0)$	$y(0)$
2	$x(2)$	$a_0x(2)$	$a_0x(1)+a_1x(0)$	$a_0x(1)+a_1x(0)$	$y(1)$
3	$x(3)$	$a_0x(3)$	$a_0x(2)+a_1x(1)$	$a_0x(2)+a_1x(1)+a_2x(0)$	$y(2)$
4	$x(4)$	$a_0x(4)$	$a_0x(3)+a_1x(2)$	$a_0x(3)+a_1x(2)+a_2x(1)$	$y(3)$

表 8-3 重定时的故障应用

时钟	输入	节点 1	节点 2	节点 3	输出
0	$x(0)$	$a_0x(0)$			
1	$x(1)$	$a_0x(1)$	$a_0x(0)$		
2	$x(2)$	$a_0x(2)$	$a_0x(1)+a_1x(0)$	$a_2x(0)$	$y(0)$
3	$x(3)$	$a_0x(3)$	$a_0x(2)+a_1x(1)$	$a_0x(1)+a_1x(0)+a_2x(0)$	
4	$x(4)$	$a_0x(4)$	$a_0x(3)+a_1x(2)$	$a_0x(2)+a_1x(1)+a_2x(1)$	

8.4.1 重定时

重定时（Leiserson 和 Saxe 1983）是一个用于在电路中改变延迟的转换技术，它并没有影响输入和输出特性。重定时已经为了时钟周期的减小（Leiserson 和 Saxe 1983）、功耗的减小（Monteiro 等 1993）及逻辑的综合被应用于同步设计中。重定时的基本过程如图 8-10 中所示，正如从 Parhi（1999）中摘录的一样。对于如图 8-10a 所示的一个有着边沿 U 和 V，以及它们之间的延迟 ω 的电路，通过如式（8-1）计算 ω'，一个重定时电路就能被推导出来，如图 8-10b 所示，延迟也变为 ω'，其中，$\boldsymbol{r}(U)$ 和 $\boldsymbol{r}(V)$ 分别是 U 和 V 的重定时值。

$$\boldsymbol{\omega}_{\mathrm{r}}(e)=\boldsymbol{\omega}(e)+\boldsymbol{r}(U)-\boldsymbol{r}(V) \tag{8-1}$$

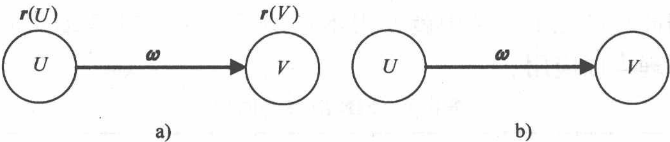


图 8-10 重定时的基本过程图

a）3 抽头 FIR 滤波器的 SFG 表示 b）移项的 SFG 表示

重定时被概括出许多的特性（Parhi 1999）：

1）任何重定时路径的权重由式（8-1）给出；

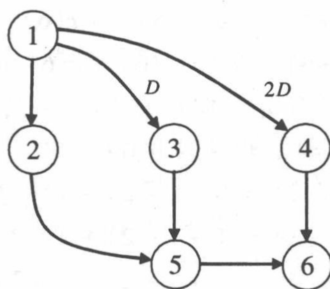
2）重定时不会改变一个周期中延迟的数量；

3）重定时不会改变一个 DFG 中的迭代界限（见后面），因为一个周期中延迟的数量不会变。

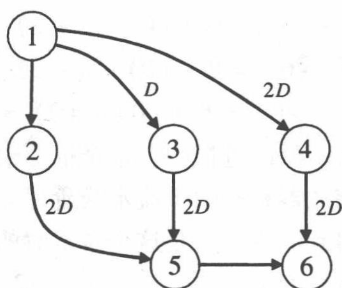
4）添加常数 j 到每个节点的重定时值上不会改变重定时图表箭头中延时的数量。

图 8-11 所示为许多应用重定时到如图 8-11a 所示的 FIR 滤波器 DFG 上的例子。为了简单，我们已经分别用 2、3、4、5，6 取代了图 8-9 的 a_0，a_1，a_2，A_0，A_1。我们已经表明了 $x(n)$ 数据资源和节点 2，3，4 之间分开的连接，这的原因会简短地提一下。通过应用式（8-1）到每个箭头上，可以得到了以下每个

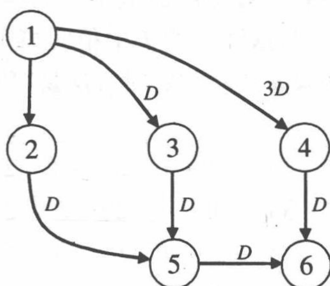
箭头的关系：



a)



b)



c)

图 8-11　重定时 FIR 滤波器

a）原始 3 抽头 FIR 滤波器的 DFG　b）重定时 FIR 滤波器的 DFG，$r(1)=-2$，$r(2)=-2$，$r(3)=-2$，$r(4)=-2$，$r(5)=0$，$r(6)=0$　c）重定时 FIR 滤波器的 DFG，$r(1)=-2$，$r(2)=-2$，$r(3)=-2$，$r(4)=-1$，$r(5)=-1$，$r(6)=0$

$$\omega_r(1\to2)=\omega(1\to2)+r(2)-r(1)$$
$$\omega_r(1\to3)=\omega(1\to3)+r(3)-r(1)$$
$$\omega_r(1\to4)=\omega(1\to4)+r(4)-r(1)$$
$$\omega_r(2\to5)=\omega(2\to5)+r(5)-r(2)$$

$$\boldsymbol{\omega}_r(3 \rightarrow 5) = \boldsymbol{\omega}(3 \rightarrow 5) + \boldsymbol{r}(5) - \boldsymbol{r}(3)$$
$$\boldsymbol{\omega}_r(4 \rightarrow 6) = \boldsymbol{\omega}(4 \rightarrow 6) + \boldsymbol{r}(6) - \boldsymbol{r}(4)$$
$$\boldsymbol{\omega}_r(5 \rightarrow 6) = \boldsymbol{\omega}(5 \rightarrow 6) + \boldsymbol{r}(6) - \boldsymbol{r}(5) \tag{8-2}$$

使用式（8-2）中的一个重定时向量 $\boldsymbol{r}(1) = -2$，$\boldsymbol{r}(2) = -2$，$\boldsymbol{r}(3) = -2$，$\boldsymbol{r}(4) = -2$，$\boldsymbol{r}(5) = 0$，$\boldsymbol{r}(6) = 0$，得到以下值：

$$\boldsymbol{\omega}_r(1 \rightarrow 2) = 0 + (-2) - (-2) = 0$$
$$\boldsymbol{\omega}_r(1 \rightarrow 3) = 1 + (-2) - (-2) = 1$$
$$\boldsymbol{\omega}_r(1 \rightarrow 4) = 2 + (-2) - (-2) = 2$$
$$\boldsymbol{\omega}_r(2 \rightarrow 5) = 0 + (0) - (-2) = 2$$
$$\boldsymbol{\omega}_r(3 \rightarrow 5) = 0 + (0) - (-2) = 2$$
$$\boldsymbol{\omega}_r(4 \rightarrow 6) = 0 + (0) - (-2) = 2$$
$$\boldsymbol{\omega}_r(5 \rightarrow 6) = 0 + (0) - (0) = 0$$

这给出了如图 8-11b 所示的改进框图，这个框图给出了一个电路，在这个电路中，输出箭头上的每个乘法器都有两个流水线延迟。在乘法器输出上提供了一个延迟的重定时向量已经得到应用，但是这个重定时的原因将在之后看到。备选的重定时向量的应用，也就是 $\boldsymbol{r}(1) = -2$，$\boldsymbol{r}(2) = -2$，$\boldsymbol{r}(3) = -2$，$\boldsymbol{r}(4) = -1$，$\boldsymbol{r}(5) = -1$，$\boldsymbol{r}(6) = 0$ 产生了如图 8-11c 所示电路，这个电路产生了一个彻底的流水线。能够从这个数字中看出对加法器流水线的应用要求一个额外的延迟 D，来应用到 1 ~4 之间的连接。从这两个例子中很明显可以看出许多重定时操作能被应用于 FIR 滤波器上。如果 $\boldsymbol{\omega}_r \geqslant 0$ 重定时方法是可行的，则在所有箭头上都可使用。

$$\boldsymbol{\omega}_r(1 \rightarrow 2) = 0 + (-2) - (-2) = 0$$
$$\boldsymbol{\omega}_r(1 \rightarrow 3) = 1 + (-2) - (-2) = 1$$
$$\boldsymbol{\omega}_r(1 \rightarrow 4) = 2 + (-1) - (-2) = 3$$
$$\boldsymbol{\omega}_r(2 \rightarrow 5) = 0 + (-1) - (-2) = 1$$
$$\boldsymbol{\omega}_r(3 \rightarrow 5) = 0 + (-1) - (-2) = 1$$
$$\boldsymbol{\omega}_r(4 \rightarrow 6) = 0 + (0) - (-1) = 1$$
$$\boldsymbol{\omega}_r(5 \rightarrow 6) = 0 + (0) - (-1) = 1$$

从两个概述的例子中很明显可以看出，重定时能被用于引入内部迭代的限制（Parhi 1999）到 DFG 中，它在最终的 FPGA 实现中作为流水线的延迟。但是，主要的议题似乎是重定时向量的决定，这个向量必定会改变 DFG 上需要的箭头的延迟，同时，$\omega_r \geqslant 0$ 保留可行的方法，也就是对所有箭头都可行的。决定重定时向量的一种方法是将图示方法对重定时符号化的 DFG 应用同步。这被称作割集或切割原理，由 Kung（1988）提出。

8.4.2　割集定理

一个 SFG（或 DFG）中的割集是箭头的最小化集，它将 SFG 分成了两部分。这个过程基于两个简单规则。

规则 1。延迟比例。所有出现在一个原始 SFG 箭头上的延迟 D 也许可以按比例表示，也就是 $D'\to\alpha D$，α 是一个正整数，它也被称作 SFG 的流水线周期。相应的，输入和输出速率也必须通过因数按比例表示（关于新的时间单元 D'）。时间比例不会改变 SFG 所有的定时。

规则 2。延迟传输（Leiserson 和 Saxe 1983）。考虑到 SFG 的任意一个将图表分成两部分的割集，依靠被分配到箭头上的方向，我们可以把割集的箭头分为入站和出站，如图 8-12 所示。延时传输规则规定大量的延迟寄存器 k，会从出站箭头转换到入站箭头，或者反之，且不会影响到整个系统的定时。

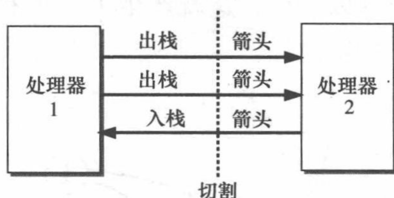


图 8-12　割集定理的应用

考虑一下将规则 2 应用到图 8-11a 中的 FIR 滤波器 DFG。在图 8-13a 中应用第一次切割，其中 DFG 图被切成两个明显的区域或子图，子图#1 包含了节点 1，2，3，4；子图#2 包含了 5 和 6。由于区间之间的所有箭头是从子图#1 出来到子图#2 的，因此延迟能被添加到每一个箭头上面，这就给出了图 8-11b。第二次切割将 DFG 分成了子图#3，包含了节点 1，2，3，5，以及子图#4，包含了节点 4 和 6。在每个箭头上添加一个单个的延迟，形成了最终的流水线设计，如图 8-11c所示。

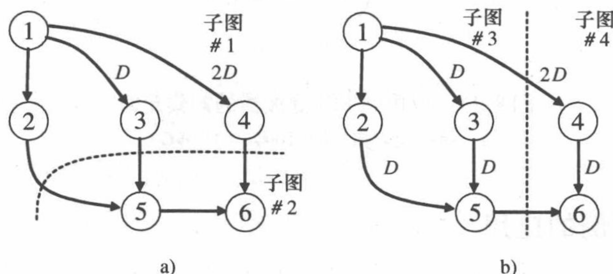


图 8-13　应用于 FIR 滤波器

a）第一次割集　b）第二次割集

这些规则提供了一个系统的添加、移除，以及分配一个 SFG 中延迟的方法，并且因此，在没有改变功能的前提下，添加、移除和分配寄存器贯穿了整个电路。然后使用割集的重定时过程来在合适的 SFG 箭头上产生足够的延迟，从而大量的延迟能从图表箭头上移除并且并入到处理模块中，这是为了在处理器中模拟流水线。如果延迟被留在箭头上，则会形成了处理器间的流水线。

当然，原始算法表示的选择会对最终的性能产生较大影响。例如采用最初在图 8-8c 及图 8-14a 中被表示为 DFG 的 SFG 的备选类型。应用割集使得乘法器的流水线能如之前一样，但是现在在节点 3 和 5 及节点 4 和 6 之间应用割集，能使得延迟得以改变，这就产生了一个有着更少数量延迟元件的电路架构，如图 8-14c所示。

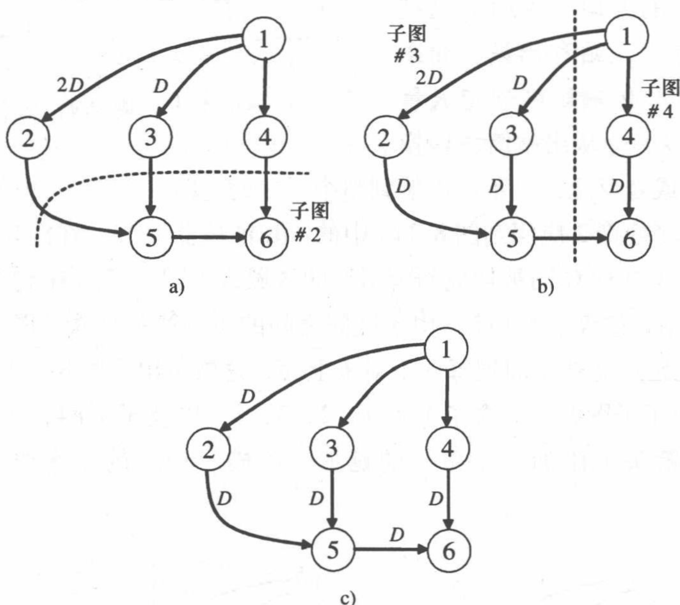


图 8-14 应用到 FIR 滤波器的割集定时

a）第一次割集 b）DFG c）DFG

8.4.3 延迟比例的应用

为了探究延迟比例方面，让我们看一个递归结构，比如式（8-3）给出的二阶 IIR 滤波器部分。方框图及相应的 DFG 如图 8-15a 和图 8-15b 所示。目的是为了在处理器层上应用流水线，因此在每个箭头上要求一个延迟 D。问题是在 2→

3→2 的循环上没有足够的延迟来应用重定时。比如，如果图形上的切割得以应用，那么这将不会改变在 3→2 到 2→3 的延迟。通过如式（8-4）和式（8-5）定义算出流水线周期并且应用时间比例，能使得这个问题得以解决。

$$y(n) = a_0x(n) + a_1x(n-1) + a_2x(n-2) + b_1y(n-1) + b_2y(n-2) \tag{8-3}$$

$$\alpha_c = B_c/D_c \tag{8-4}$$

$$\alpha = \max\{all\alpha_c\} \tag{8-5}$$

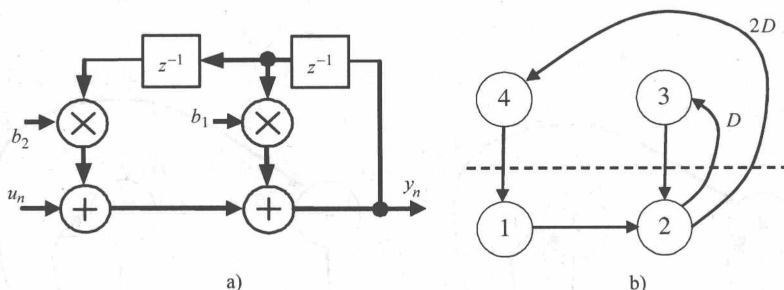


图 8-15　二阶 IIR 滤波器

a）模块框图　b）DFG

在式（8-4）中，值 B_c 指的是对处理器流水线要求的延迟，值 D_c 指的是在原始 DFG 中可用的延迟。最优化流水线周期通过使用式（8-5）得以计算，然后被用作比例因数。正如所显示的，有两个循环，循环限制#2 是最糟糕。这些循环是根据单元时间（u. t.）步长给出的。

1→2→4→1(3u. t.)

2→3→2(2u. t.)

循环限制#1(3/2 = 1.5u. t.)

循环限制#2(2/1 = 2u. t.)

应用比例和重定时的过程如图 8-16 所示。应用 2 为比例，给出如图 8-16a 所示的重定时 DFG。应用图中的切割产生了如图 8-16b 所示的修改的 DFG，此外，其中有如图 8-16c 所示的 DFG 的另一个切割应用。将延迟映射到处理器并添加数字来显示流水线层产生了最终的流水线 IIR 递归式，如图 8-16d 所示。

最后的实现通过使用 Xilinx Virtex 5 FPGA 得以综合并且综合的结果可通过表 8-4 中的图 8-15b 和图 8-16d 的电路观察到。

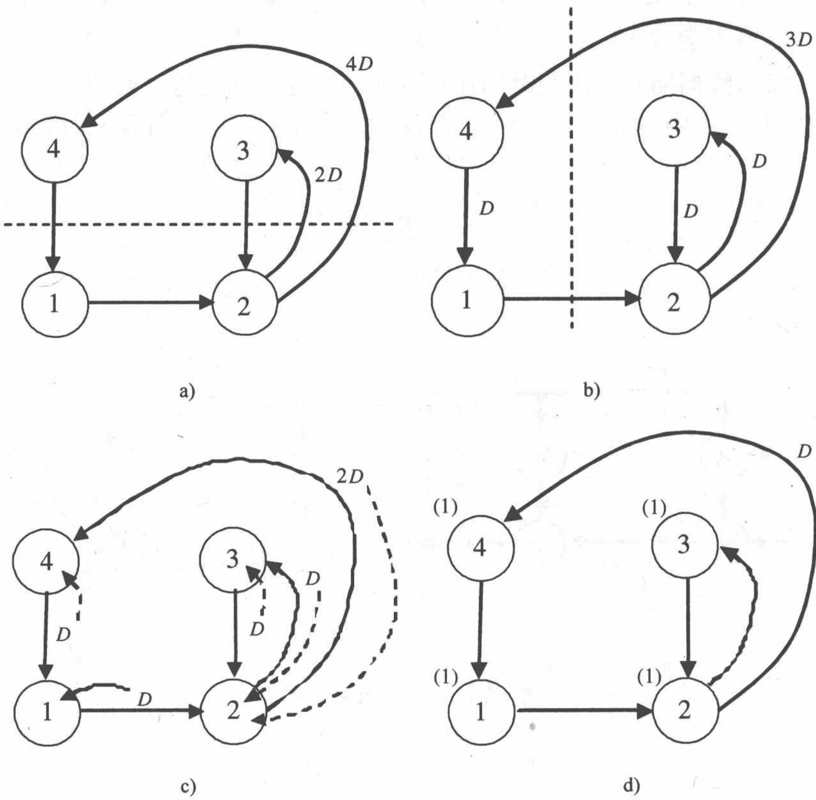


图 8-16 二阶 IIR 滤波器的流水线

a）缩放 IIR 递归 b）递归第二切割应用 c）移动到处理器的延迟 d）最终的流水线 IIR 递归

表 8-4 重定时的有错误的应用

面积			吞吐量	
电路	DSP48	触发器	时钟（MHz）	数据速率（MHz）
图 8-15b	2	20	176	176
图 8-16d	2	82	377	188

8.4.4 流水线周期的计算

先前的章节已经概述了第一个决定流水线周期的过程，使得这个流水线周期能够通过放缩在处理器层使用，在 FPGA 技术中，在这一层使用流水线可能是最好的（虽然，正如将在第 14 章中见到的，添加更高层的流水线有利于低功耗 FPGA 的实现）。但是，流水线周期的计算仅仅实施在一个 IIR 滤波器二阶的部

分简单例子上，并且因此需要更多的计算流水线周期的方法。已经在 Parhi (1999) 中已经提出了许多的不同的技术。在这里要考虑的一点就是最长路径矩阵算法（Parhi 1999），并且用一个例子很好地阐述了。

最长路径矩阵算法

构建了一系列矩阵并且通过检测对角线元素发现了迭代界限。如果 d 是 DFG 中延迟的数量，则创建一个 $\boldsymbol{L}^{(m)}$，$m=1, 2, \cdots, d$ 的矩阵，这样元素 $l_{1,j}^{1}$ 是从穿过 $m-1$ 个延迟的延迟元件出来的最长路径（不包括 d_i 和 d_j）。如果没有路径存在，则 $l_{1,j}^{1}$ 就是 -1。通过使用 Bellman-Ford 或 Floyd-Warshall 算法（Parhi 1999）可以计算出最长路径。

例 1　考虑图 8-17 中给出的例子。由于目的是产生一个流水线型的电路，因此从每个处理器中的（1）表达式指出的流水线类型入手。如果流水线并不必要，那么这经由将表达式改为（0）得以实现，或者如果在路由中需要额外的流水线延迟来帮助布置和路由或为了低功耗的实现，那就需要将其改变为比如（2）或（3）等的表达式来得以实现。

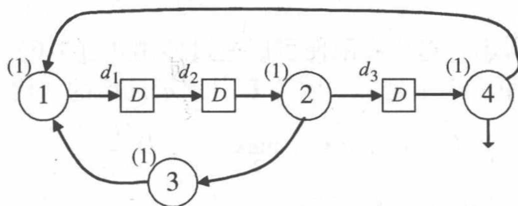


图 8-17　简单 DFG 举例（Parhi 1999）

第一阶段是计算矩阵 $\boldsymbol{L}^{(m)}$，从矩阵 $\boldsymbol{L}^{(m)}$ 入手。通过产生每一个命名为 $l_{1,j}^{1}$ 计算出来的，它由 d_i 到 d_j 延迟的路径得出。比如，从 d_1 回到 d_1 路径为 1（$d_1 \to d_2 \to 2 \to 3 \to 1 \to d_1$），得到延迟为 2（$d_1 \to d_2 \to 2 \to d_4 \to 1 \to d_1$），因此（1, 1）$= -1$。对于 $l_{3,1}^{1}$，是由 d_3 到 d_1 的路径，穿过节点（4）和（1），得到延迟为 2；因此，$l_{3,1}^{1}=2$。对于 $l_{2,1}^{1}$，是由 d_2 到 d_1 的路径，穿过节点（2），（3）和（1），因此 $l_{2,1}^{1}=3$。这给出了如下的矩阵：

$$\begin{pmatrix} -1 & 0 & -1 \\ 7 & -1 & 3 \\ 3 & -1 & -1 \end{pmatrix}$$

更高阶的矩阵不需要从 DFG 推导。它们能被递归地计算如下：

$$l_{i,j}^{m+1} = \overset{\max}{k\varepsilon K}(-1, l_{i,j}^{1} + l_{k,j}^{m})$$

式中，K 是整数 k 在区间 $[1, d]$ 之间的集合，以至于既没有 $l_{i,k}^{1} = -1$ 也没有

$l_{i,k}^{m}=-1$。为了满足 $l_{1,1}^{2}$，我们考虑 $K=1$，2，3，但是1，3 包括了 -1，因此仅有 $K=2$ 是有效的。因此

$$l_{1,1}^{2}=\overset{\max}{k\varepsilon}3(-1,0+7)$$

整个 $\boldsymbol{L}^{(2)}$ 通过以下方式产生：

$$\underset{\boldsymbol{L}^{(1)}}{\begin{pmatrix}-1 & 0 & -1\\ 7 & -1 & 3\\ 3 & -1 & -1\end{pmatrix}}\underset{\boldsymbol{L}^{(1)}}{\begin{pmatrix}-1 & 0 & -1\\ 7 & -1 & 3\\ 3 & -1 & -1\end{pmatrix}}\Rightarrow\underset{\boldsymbol{L}^{(2)}}{\begin{pmatrix}7 & -1 & 3\\ 6 & 7 & -1\\ -1 & 3 & -1\end{pmatrix}}$$

当 $\boldsymbol{L}^{(2)}$ 仅通过使用 $\boldsymbol{L}^{(1)}$ 计算时，矩阵 $\boldsymbol{L}^{(3)}$ 通过使用如下所示的 $\boldsymbol{L}^{(1)}$ 和 $\boldsymbol{L}^{(3)}$ 计算，对每个元件的计算和之前一样如下所示：

$$l_{i,j}^{3}=\overset{\max}{k\varepsilon}K(-1,l_{i,j}^{1}+l_{k,j}^{2})$$

这产生了 $\boldsymbol{L}^{(3)}$ 的计算如下：

$$\underset{\boldsymbol{L}^{(1)}}{\begin{pmatrix}-1 & 0 & -1\\ 7 & -1 & 3\\ 3 & -1 & -1\end{pmatrix}}\underset{\boldsymbol{L}^{(2)}}{\begin{pmatrix}7 & -1 & 3\\ 6 & 7 & -1\\ -1 & 3 & -1\end{pmatrix}}\Rightarrow\underset{\boldsymbol{L}^{(3)}}{\begin{pmatrix}6 & 7 & -1\\ 14 & 6 & 10\\ 10 & -1 & 6\end{pmatrix}}$$

一旦矩阵 $\boldsymbol{L}^{(m)}$ 得出，迭代界限便可以通过使用式（8-6）决定。在这种情况下，$m=3$ 是因为这里有 3 个延迟，因此 $\boldsymbol{L}^{(3)}$ 表示最后的迭代。

$$T_{\infty}=i,\ m\varepsilon1,\ \underset{2}{\max},\ \cdots,\ D\left\{\frac{l_{l,l}^{m}}{m}\right\} \tag{8-6}$$

对于这个例子，给出了如下式子：

$$T_{\infty}=\left\{\frac{7}{2},\frac{7}{2},\frac{6}{3},\frac{6}{3},\frac{6}{3}\right\}=4$$

例 2 考虑如图 8-18a 所示的网格滤波器 DFG 结构。通过为每个处理器选择一个单个的延时（1），又一次选择了一个流水线的类型。

4 个可能的矩阵值决定如下：

$D_1\to M_3\to A_3\to D_1$

$D_1\to A_4\to D_2$ 及 $D_1\to M_4\to A_3\to M_3\to A_4\to D_2$

$D_2\to M_2\to A_1\to A_3\to D_1$

$D_2\to M_2\to A_1\to A_3\to M_2\to A_4\to D_2$

因此给出

$$\begin{pmatrix}2 & 4\\ 3 & 5\end{pmatrix}$$

更高阶的矩阵 $\boldsymbol{L}^{2}$ 计算如下：

$$\underset{\boldsymbol{L}(1)}{\begin{pmatrix}2 & 4\\ 3 & 5\end{pmatrix}}\underset{\boldsymbol{L}(1)}{\begin{pmatrix}2 & 4\\ 3 & 5\end{pmatrix}}\Rightarrow\underset{\boldsymbol{L}(2)}{\begin{pmatrix}7 & 9\\ 8 & 10\end{pmatrix}}$$

这给出了如下的迭代界限：

$$T_{\infty} = \overset{\max}{i,m\varepsilon 1,2}\left\{\frac{l_{i,i}^{m}}{m}\right\} \tag{8-7}$$

它解决了

$$T_{\infty} = \left\{\frac{2}{1},\frac{5}{1},\frac{7}{2},\frac{10}{2}\right\} = 5$$

将这个比例因数应用到如图 8-18b 所示的网格滤波器 DFG 结构中，得出了最后如图 8-18c 所示的结构，它有流水线式的处理器，正如添加到每个处理器上的（1）表达式指出的。这个最后电路的得出是依靠跨不同割集的延迟应用和在处理器层重定时的应用，这样可以将延迟从输入传递到输出。

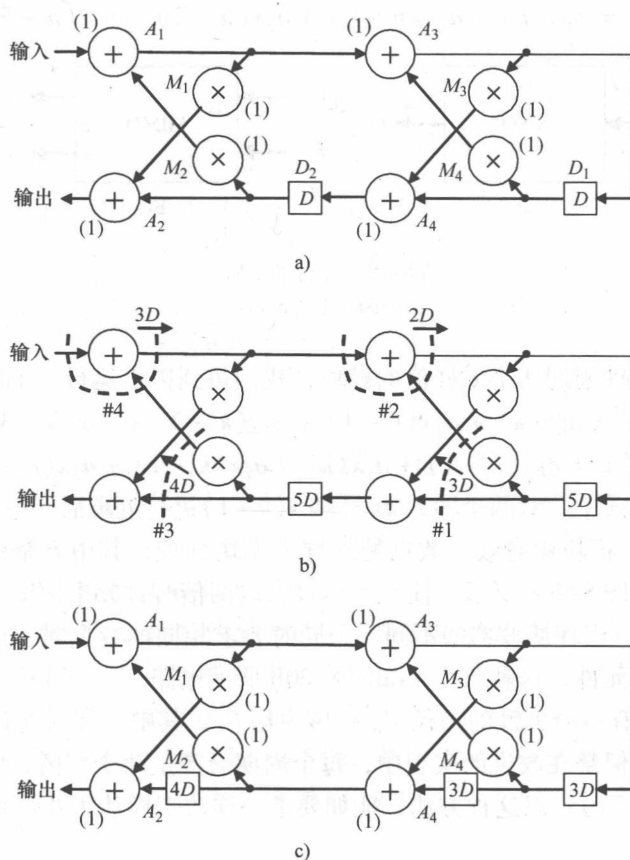


图 8-18　网格滤波器（Parhi 1999）

a）DFG　b）表明了切割的 DFG　c）重定时 DFG

8.5 并行运算

先前的章节用非常卓越的方法来将流水线应用到一个现有的 DFG 表示上，主要是基于处理器层流水线的应用，因为这一层是 FPGA 中能最好地使用流水线的一层。这会对要求提升速度这一原则产生影响，正如表 8-4 中的结果所证明的，并且 FIR 滤波器更能明显地提升速度。另一个提高性能的方法是提高硬件的并行化，如图 8-19 所示。这是通过将如图 8-19 所示的 SISO 系统转变到一个如图 8-19b 所示的 MIMO 系统做到的。

考虑早先给出的简单的 FIR 滤波器。接下来考虑给出的 4 抽头延时线性滤波器。

$$y(n) = a_0x(n) + a_1x(n-1) + a_2x(n-2) + a_3x(n-3) \tag{8-8}$$

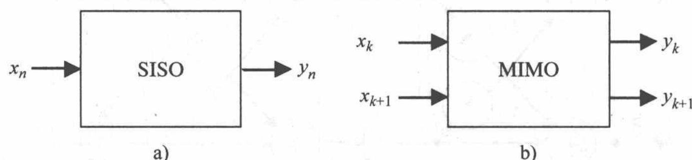


图 8-19 并行的处理

a）SISO b）MIMO

假定每个时钟周期两个采样值的模块，我们得到以下运行一个周期的迭代。

$$y(k) = a_0x(k) + a_1x(k-1) + a_2x(k-2) + a_3x(k-3)$$

$$y(k+1) = a_0x(k+1) + a_1x(k) + a_2x(k-1) + a_3x(k-2)$$

在以上表达式中，对两个输入 $x(k)$ 和 $x(k+1)$ 进行处理后产生相应的输出 $y(k)$ 和 $y(k+1)$。模块中有效的数据处理称为模块处理，其中 k 是模块大小。两个周期的框图如图 8-20a 所示。注意到当数据以两倍时钟频率被馈送时，在这些结构中延迟为 k。当在滤波器的不同部分同时要求相同的数据时，能利用这一点来减少一些延迟元件，这就产生了如图 8-20b 所示电路。

FIR 滤波器有一个关键的路径 $T_M + (N-1)T_A$，其中 N 是决定时钟周期的滤波器抽头数量。但是在改进的实现中，每个周期会产生两个样值，因此吞吐量为 $2/(T_M + (N-1)T_A)$。以这种方法，能如要求一样改变模块大小，但是这增加了硬件成本。

Parhi（Parhi 1999）引进了一项技术，在这项技术中，计算量通过如下重排计算得以减小。

$$y(k) = a_0x(k) + a_2x(k-2) + z^{-1}(a_1x(k+1) + a_3x(k-1))$$

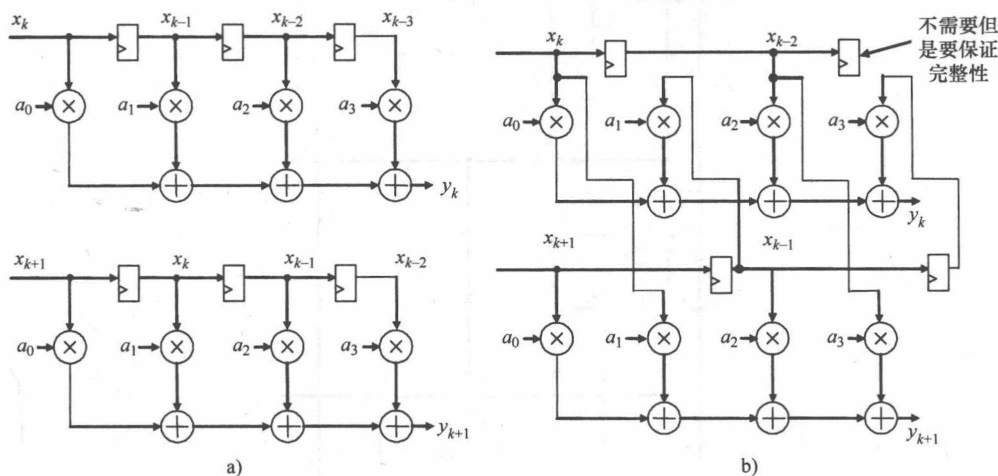


图 8-20　FIR 过滤模块
a）两次迭代　b）复合运算

通过创建如下的两抽头滤波器，$y(1k) = a_0x(k) + a_2x(k-2)$ 和 $y(2k) = a_1x(k+1) + a_3x(k-1)$，我们如下重算表达式 $y(k)$。

$$y(k) = y(1k) + z^{-1}(y(2(k+1)))$$

表达式 $y(k+1)$ 被重写为如下：

$$y(k+1) = (a_0 + a_1)(x(k+1) + x(k)) + (a_2 + a_3)(x(k-2)) - a_0x(k) - a_1x(k+1) - a_2x(k-2) - a_3x(k-1)$$

这产生了一个如下的一个 2 抽头滤波器，包含了一个系数为（$a_0 + a_1$）和（$a_2 + a_3$）的结构。因此减小了原始 4 抽头滤波器的复杂度。它包括两方面的减少，也就是 $y(k)$ 和 $y(2k+1)$，但是这些早就由 $y(k)$ 的计算得出了。产生的影响是以一个加法/减法的代价减少两个乘法运算。这对一个 FPGA 的实现来说却不一定重要，在实现中乘法运算的复杂度可与标准字长的加法相比。更重要的是，顶层和底层滤波器在长度上被减小了 $2(N/2)$ 个抽头并且产生了一个额外的 $2(N/2)$ 抽头滤波器来实现每个表达式中的第一行。一般来说，滤波器已经被减半，因此关键路径为 $T_M + (N/2)T_A + 3T_A$，其中包含了 3 个加法器，其中一个滤波器计算为 $x(k) + x(k+1)$，一个减去 $y(1k)$，另一个减去 $y(2(k+1))$，如图 8-21 所示。

$$y(k+1) = (a_0 + a_1)(x(k+1) + x(k)) + (a_2 + a_3)(x(k-2)) + x(k-2)$$
$$y(1k) - y(2(k-1))$$

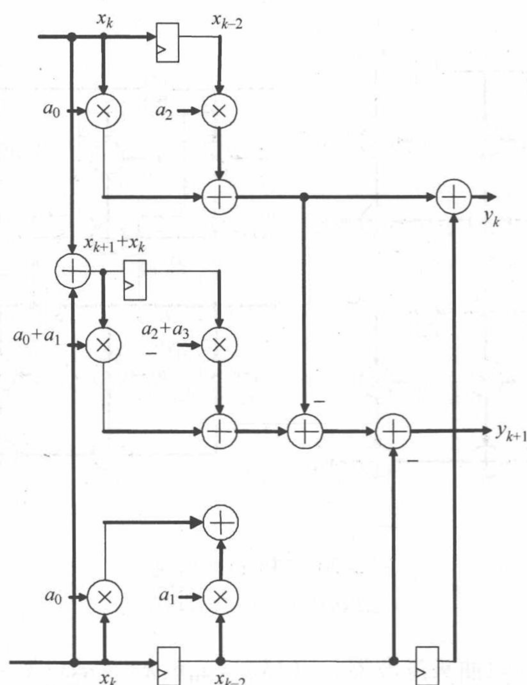


图 8-21 减少的基于模块的 FIR 滤波器

8.6 硬件共享

8.6.1 不折叠

先前的章节表明了我们如何能在模块中执行并行计算，严格地说这称为不折叠。不折叠是一种转换技术，它能被应用到一个 DSP 程序中来产生一个新的程序，它比原始程序的一个迭代的效率更高。通过使用一个描述迭代次数的不折叠因数 J 得以表述。例如，考虑展开一阶 IIR 滤波器这部分，$y(n)=x(n)+by(n-1)$迭代 3 次，表达式如下：

$$y(k) = x(k) + by(k-1)$$
$$y(k+1) = x(k+1) + by(k)$$
$$y(k+2) = x(k+2) + by(k+1)$$

SFG 和 DFG 表示如图 8-22a 所示，在里面加法器被处理器 A 代替，乘法器被 B 代替。在图 8-22b 中给出了不折叠的类型，其中 A_0，A_1，A_2 表示计算 3 个加法的硬件。B_0，B_1，B_2 表示计算 3 个乘法的硬件。有了非循环的表达式，每

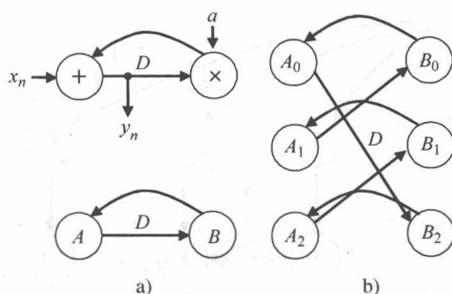


图8-22　不折叠的一阶递归
a）SFG和DFG　b）不折叠操作

个延迟现在都等于3个时钟周期。例如，先前在处理器上需要的值 B_0 是 $y(n-1)$，它是通过 A_2 输出也就是 $y(n+2)$ 的3个延迟产生的。当与原始的SFG相比时，延迟似乎已经在不同的弧度，例如 A_0-B_0，A_1-B_1 和 A_2-B_2 之间被重新分配了。

Parhi（Parhi 1999）给出了一个自动执行不折叠的算法并且在此得以引用。这是基于一个事实的，这个事实是在不折叠因数为 J 的不折叠DFG中，节点 $U(i)$ 的第 k 次迭代是执行原始DFG中节点 U 的第 $J(k+i)$ 次迭代。

不折叠算法步骤：

1）对在原始DFG中的每个节点 U，画出 J 个节点 $U(0)$、$U(1)$、…、$U(J-1)$；

2）对每个在原始DFG中有着延迟 w 的 $U\rightarrow V$，J 边缘 $U(i)\rightarrow V(i+w)/J$，延迟为 $(i+w\%J)$，边缘为 $i=0,1,\cdots,J-1$，其中%是求余。

考虑FIR滤波器DFG，作为如图8-23a所示的FIR滤波器框图的DFG表示。在变换图中，新的箭头计算及各种延迟的计算，在以下给出。这产生了如图8-23b所示的不折叠DFG，它等同于图8-23a中给出的折叠电路。

$$X_0\rightarrow A(0+0)\%2=A(0)\text{，延迟}=[0/2]=0$$
$$X_1\rightarrow A(1+0)\%2=A(1)\text{，延迟}=[1/2]=0$$
$$X_0\rightarrow B(0+1)\%2=B(1)\text{，延迟}=[1/2]=0$$
$$X_1\rightarrow B(1+1)\%2=B(2)\text{，延迟}=[2/2]=1$$
$$X_0\rightarrow C(0+0)\%2=C(0)\text{，延迟}=[2/2]=1$$
$$X_1\rightarrow C(1+2)\%2=C(1)\text{，延迟}=[3/2]=1$$
$$X_0\rightarrow D(0+3)\%2=D(1)\text{，延迟}=[3/2]=1$$
$$X_1\rightarrow D(1+3)\%2=D(0)\text{，延迟}=[4/2]=2$$

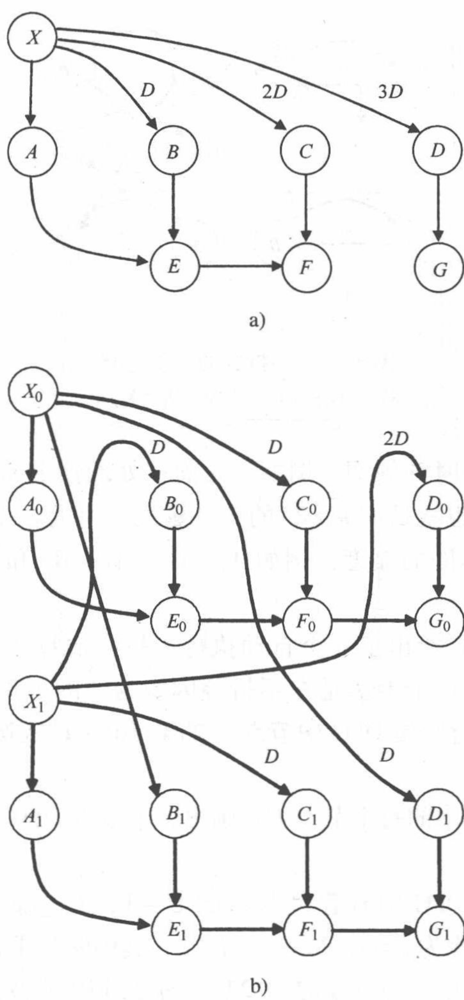


图8-23 不折叠的FIR滤波器的模块

a）SFG和DFG b）不折叠操作

8.6.2 折叠

先前的章节概述了FIR滤波器结构并行实现的一种技术。但是在一些情况下，需要的是执行硬件共享或通过一个因数 k 的硬件折叠来减小硬件的数量，并也因此减小采样速率。考虑如图8-24a所示的FIR滤波器框图。通过将滤波器结构折叠4次，就推导出了如图8-24b所示电路。在改进的电路中，随着操作计划映射到单个硬件单元上，硬件要求已经减少了4个，见表8-5。

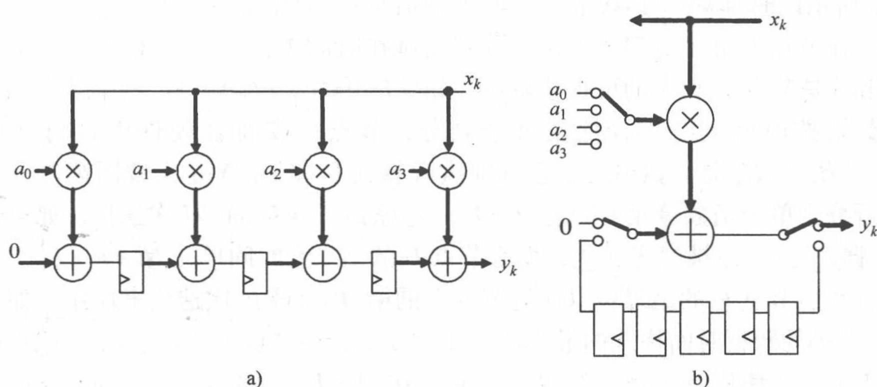


图 8-24 折叠的 FIR 滤波器部分

a）原始框图模块 b）折叠电路（采样率 = 1/4）

表 8-5 图 8-24b 的调度

时钟周期	加法器输入	加法器输入	加法器输出	系统输出
0	a_3	0	$a_3x(0)$	$(y(3)''')$
1	a_2	0	$a_2x(0)$	$(y(2)'')$
2	a_1	0	$a_1x(0)$	$(y(1)')$
3	a_0	0	$a_0x(0)$	$y(0)$
4	a_3	0	$a_3x(1)$	$(y(4)''')$
5	a_2	$a_2x(1)$	$a_2x(1)+a_3x(0)$	$(y(3)'')$
6	a_1	$a_1x(1)$	$a_1x(1)+a_2x(0)$	$(y(2)')$
7	a_0	$a_0x(1)$	$a_1x(1)+a_2x(0)$	$y(1)$
8	a_3	0	$a_3x(2)$	$(y(5)''')$
9	a_2	$a_2x(1)+a_0x(0)$	$a_2x(1)+a_2x(1)+a_3x(0)$	$(y(4)'')$

根据周期数，数据的时序被分别标为 0，1，2，3，每 4 个周期重复一次（严格来说，应该是 k，$k+1$，$k+2$ 和 $k+3$）。从表格中很明显可以看出，结果是每 4 个周期产生一次的，也就是在第 4 个、第 8 个周期，以此类推。部分结果在括号中给出了，因为它们不会作为一个完整输出产生。表达式 $y(3)'''$表示第一部分 $y(3)$ 和 $y(3)''$的产生，$y(3)$表示的第二部分以此类推。

这个折叠的等式（Parhi 1999）由式（8-9）给出，其中单个组件的所有输入同时到达，并且从每个输入到输出的流水线层是相同的。

$$D_F(U \xrightarrow{e} V) = Nw(e) - P_u + v - u \tag{8-9}$$

式中，$w(e)$是在 $U \xrightarrow{e} V$ 里延迟的数量；N 是流水线周期；P_u 是 H_u 输出引脚的流水线站；u 和 v 是节点 U 和 V 的可折叠阶数，满足 $0 \leqslant u$，$v \leqslant N-1$。考虑如图

8-25a 所示以时延 $w(e)$ 连接节点 U 和 V 的箭头 e，其中节点 U 和 V 也许是分层的模块。让节点 U 和 V 的第 l 次迭代分别安排在时间单元 $NL+u$ 和 $NL+v$ 上，其中 u 和 v 是节点 U 和 V 的可折叠阶数，并满足 $0 \leqslant u$，$v \leqslant N-1$。一个节点的可折叠阶数是按时间划分的，依据这个划分，节点被安排在硬件中执行（Parhi 1999）。H_u 和 H_v 是函数单元，它分别执行节点 U 和 V。N 是折叠因数并且被定义为折叠到单个函数单元上的运行次数。考虑到节点 U 的第 l 次迭代。如果输出 H_u 引脚通过 P_u 站被流水线化，那么节点 U 的结果在时间单元 $Nl+u+P_u$ 上是可用的，并且节点 U 的结果可以通过节点 V 的第 $(l+w(e))$ 次迭代来使用。如果输入 H_u 引脚的数据时间格式的最小值是 A_v，那么能在 $N(l+w(e))+v+A_v$ 上执行节点 V 的输入引脚。因此，结果必须储存在 $D_F(U \xrightarrow{e} V) = [N(l+w(e))+v+A_v] - [Nl+P_u+A_u+u]$ 单元里。路径从 H_u 到 H_v 需要 D'_F（$U \xrightarrow{e} V$）延时，并且在这个路径上的数据是在 $Nl+v+A_v$ 上的输入 H_v，如图 8-25b 所示。因此，在式（8-10）中给出了分层的复杂成分的折叠等式。

$$D_F(U \xrightarrow{e} V) = Nw(e) - P_u + A_v + v - u \tag{8-10}$$

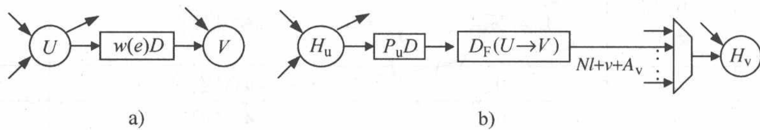


图 8-25 折叠转换

a) 时延为 $w(e)$ 的 $U \to V$ 箭头 b) 相应的折叠数据通道

这个表达式能被系统地应用到如图 8-24a 所示的框图中来推导出如图 8-24b 所示电路。为了便于展示，使用了如图 8-26a 所示的 DFG。在图中，为了折叠的简化，已经添加了一个额外的加法器 H。在图 8-26a 中，我们已经使用大量括号来表明处理原件所需的阶数。因此，这个目的表明了我们想要使用一个加法器来计算 $a_3x(n)$，$a_2x(n)$，$a_1x(n)$ 和 $a_0x(n)$。因此，这些时序指明了设置的阶数 u 和 v 的值。下一个计算如下得出，它给出了如图 8-26a 所示的时延和时序要求。

$$D_{F(A \to H)} = 4(0) - 0 + 1 - 1 = 0$$
$$D_{F(B \to E)} = 4(0) - 0 + 2 - 2 = 0$$
$$D_{F(C \to F)} = 4(0) - 0 + 3 - 3 = 0$$
$$D_{F(D \to G)} = 4(0) - 0 + 4 - 4 = 0$$
$$D_{F(H \to E)} = 4(1) - 0 + 2 - 1 = 5$$
$$D_{F(E \to F)} = 4(1) - 0 + 3 - 2 = 5$$
$$D_{F(F \to G)} = 4(1) - 0 + 4 - 3 = 5$$

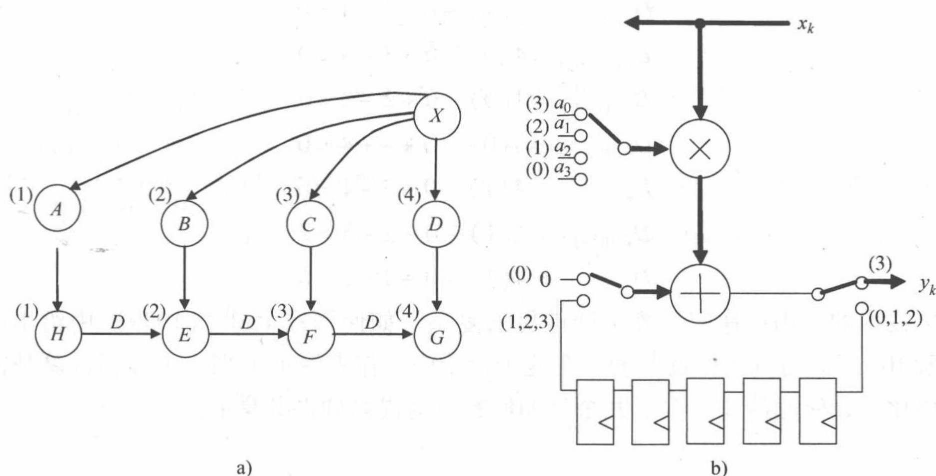


图 8-26　折叠的进程

a）原始框图　b）折叠电路（采样速率 = 1/4）

图 8-27a 所示为反转时间序列是如何产生一个稍有不同的折叠电路（图 8-27b）的，其中已经改变了反馈循环上的延迟，也相应改变了数据选择器上的时序。这个例子演示了改变计算上时间排序的影响。在以下显示了不同的时序计算。以下的例子是一组序数的操作，分别变为（1），（3），（2）和（4）。并且要求加法器输出和加法器输入之间要有两个不同的连接，并且延迟也不一样，分别是 3 和 6。

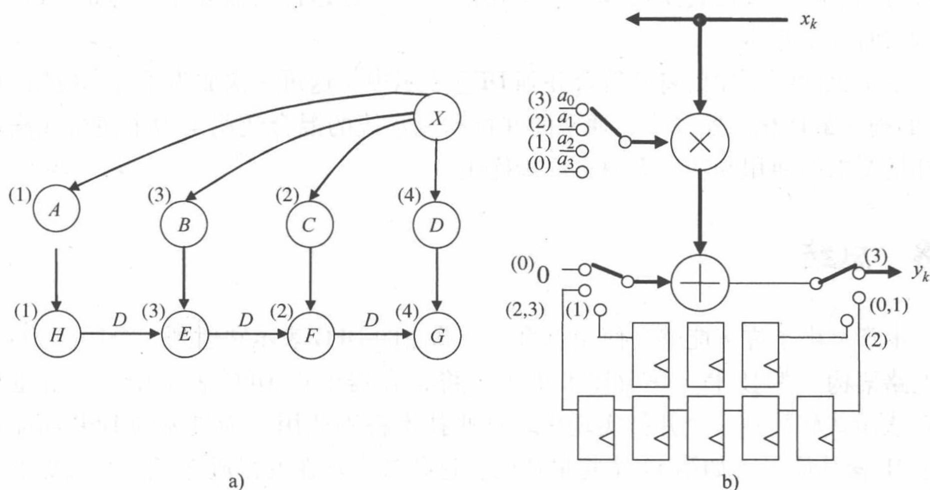


图 8-27　备选的折叠

a）改进的框图　b）改变了调度表的电路

$$D_{F(A\to H)}=4(0)-0+1-1=0$$
$$D_{F(B\to E)}=4(0)-0+3-3=0$$
$$D_{F(C\to F)}=4(0)-0+2-2=0$$
$$D_{F(D\to G)}=4(0)-0+4-4=0$$
$$D_{F(H\to E)}=4(1)-0+3-1=6$$
$$D_{F(E\to F)}=4(1)-0+2-3=3$$
$$D_{F(F\to G)}=4(1)-0+4-2=6$$

技术的应用在递归计算中变得更加复杂，如使用在 Parhi（1999）中给出的二阶 IIR 滤波器例子所证明的。在这个例子中，作者演示了当一个递归计算被流水线化时，研究涉及的自然冗余是如何有效地提高硬件共享的。

8.7 FPGA 中的应用

本章已经简洁地概括了将算法描述以 DFG 的形式映射到电路架构中的一些技术。最初的资料展示了我们如何能应用延迟比例来首先将足够的延迟添加到 DFG 中，这是为了使重定时得以应用。这转换到了 FPGA 的实现上，其中寄存器的数量能随要求变化。正如第 5 章所描述的，流水线的使用是产生高性能 FPGA 实现的一个有力技术。

在提出的这个设计例子中，选择了一个一重流水线，因为这代表了 FPGA 中可能最好的流水线重数。这能通过确保箭头上内部迭代的界限做到，这个迭代能被映射到节点中来代表流水线。保留在箭头上的延迟代表需要用于保证 DFG 正确重定时的寄存器。

本章也回顾了如何将并行合并到 DFG 表示中，这再一次成为了一个应用 FPGA 的现实最优化。实际上，通常的并行和流水线的混合使用是为了使得在满足吞吐量要求的面积和功率方面实现最优化。

8.8 总结

本章突出了许多能够让使用者将一个算法的 DFG 表示映射到一个适合 FPGA 的电路架构中的技术。这项技术集中于将寄存器引进 DFG 表示中，并且也将 DFG 表示式转换到一个并行实现中。这些技术特别适用于为具体的 DSP 功能性产生 IP 核功能。正如第 11 章将描述的，这些技术现在变得成熟了，并且焦点从高级描述转换为产生有效率的系统实现上，在高级描述中，节点功能也许已经以 IP 核的形式形成了。因此，本书剩余的部分将集中在这个更高级的问题上。

参考文献

Leiserson CE and Saxe JB (1983) Optimizing synchronous circuitry by retiming *Proc. Third Caltech Conference on VLSI*, pp. 87–116.

Kung SY (1988) *VLSI Array Processors*. Prentice Hall, Englewood Cliffs, NJ.

Monteiro J, Devadas S and Ghosh A (1993) Retiming sequential circuits for low power. *Proc. IEEE Int. Conf. on Computer Aided Design*, pp. 398–402.

Parhi KK (1999) *VLSI digital signal processing systems : design and implementation*. John Wiley and Sons, Inc., New York.

第 9 章 IRIS 行为综合工具

第 7 章已经强调了基于 FPGA 的 DSP 系统的哪些工具和方法是可用的，特别强调了综合专用 IP 核的工具。但是，第 8 章探讨了在建立电路架构中的一些议题，而这些是 IP 核的基础。像在 DFG 的算法表达式中，对并行性和流水线型式的算法并发性的探究有助于 FPGA 实现的产生。虽然使用了大量简单的例子来叙述基础的技术，但是现实是这些技术很难对不怎么复杂的算法进行研究。由于这个原因，研发综合工具来使这些技术自动化会有很多的收益。

行为综合工具可以接受对硬件有要求的功能行为的描述，然后从由工具供应商或使用者提供的元件库中选择合适的硬件组件，产生组件间需要的互连，将工作和在行为描述中使用的数据分配到用来计算和存储的元件中，确定出在计算和存储的元件上执行的操作顺序，然后产生出实现这个连续操作的一个规范。通过使用逻辑综合工具，针对 FPGA 的实现来产生出最后的设计。为了更有效率，行为综合工具必须满足使用者对综合架构的指定限制以及/或者优化目标。这些主要包括成本、时钟周期、吞吐量和时延因素。这些工具也应该为综合进程的结果提供直观的展示，这种直观展示使得使用者能快速理解综合的架构，以及理解分配和调度的结果。本章将描述已经在 Queen's University 得到了开发的 IRIS 综合工具，并且实际上已经实现了在第 8 章中研究的技术。这个工具代表了大量行为综合的一个例子，它从 SFG 角度给出了实现基于 FPGA 的 DSP 系统的细节。本章是对 Yi 和 Woods（2006）提出的论文的扩展。

本章由以下组成。9.1 节将给出一个对行为综合工具的简单介绍，其目的是为了强调一些在本章中描述的过程。9.2 节将描述 IRIS 综合工具，它是基于模块化设计过程的，它将在 9.2.1 节中描述。综合过程的一个关键方面是重定时，这在 9.3 节中有描述。第 8 章中的例子是简单的 DFG 描述，但是在 9.4 节中的描述将展示在产生 SFG 功能性分级实现中的挑战。9.5 节将继续描述硬件共享如何在分级的描述功能中得到实现。

9.1 行为综合工具的介绍

一些供应商为数字系统综合提供 CAD 工具，从 RTL 级入手，主要是使用综合的某一形式。存在许多基于 FPGA 的 DSP 设计的成熟的低级设计工具，比如 Synplify（Synplify 2003）和 Design Manager（Xilinx Inc. 2001）。但是可以很清楚

地看出，还需要高级综合工具来有效率地实现系统和架构级的综合。在行为层或算法层上，规格和算法一样，在这个算法中，基础结构的元素是控制器和网表。架构级综合（Vanhoof 等 1993）从一个行为的（算法的）规范入手，产生 RTL 层结构的系统描述。

我们使用以下的度量来定义一个好的综合系统（Roy 1993）。

（1）综合复杂设计的能力。一个好的综合系统应该能够处理有合理规模和复杂度的例子。

（2）更短的设计周期。一个优秀的综合系统可以减小产品生命周期并且大大降低芯片成本。

（3）广泛的设计空间的探索。综合系统有必要探索广阔的设计空间来产生或接近最佳的设计。

（4）现实约束驱动的综合。我们应该考虑的是现实的约束条件，而不是抽象的。

但是以上的一些度量是相互冲突的，并且在一个单独的处理器实现中，在其中一个度量中的提高可能会导致另一方面的下滑。例如，广泛的设计空间探索通常会减缓设计进程，那意味着设计周期会增加。各种与架构极综合相关的任务包括安排一些运算符来控制步骤（调度）、将这些运算符分配到描述中的操作上（分配）、给一些操作配置一些运算符（捆绑），以及给变量分配和配置存储器模块（配置和捆绑）。

调度、资源分配和捆绑是涉及在设计空间中找到一个满意解的任务。调度是在综合进程中一个重要的子任务。调度会影响综合设计的几个方面，包括使用的功能性单元的总数、计算的总时间、存储和互连的要求。这里主要的焦点是最小化功能单元的数量，从而减小电路规模。调度算法能被清晰地分为时间约束和资源约束。在时间约束的调度中，为了控制步骤的数量固定，功能单元的数量被最小化。在资源约束的调度中，控制步骤的数量是因为设计成本最小化，也就是功能性和存储单元的数量。先前的综合方法包括尽可能快（As Soon As Possible，ASAP），或者尽可能晚（As Late As Possible，ALAP）的调度，这是解决有着优先级约束（Dewilde 等 1985）调度问题的一个最简单的方法。ASAP 和 ALAP 调度有着共同的缺点，那就是算法的任何地方都没有涉及资源的使用。因此，没有最小化成本和考虑资源约束的尝试。列表调度方法（Davidson 等 1981）通过在移动到下一步前，和瞬间执行大量操作一样，在递增序数和调度中识别出可用的瞬间来针对资源约束的问题。整数线性规划（Integer Linear Programming，ILP）方法（Lee 等 1989）试图使用分枝定界搜索算法来找到一个最佳调度，它是相对简单的自动化。大多数这些算法都是基于一个单形法的。ILP 方程的复杂度随着控制步骤的数量迅速地提升。实际上，ILP 方法仅仅对非常小规模的问题是可

应用的。

强制定向调度的方法（Force Directed Scheduling，FDS，Paulin 和 Knight 1989）是通过平衡操作、存储的值及数据传输的并发性来使得受给定时间限制的硬件面积最小化。迭代优化（Iterative Refinement，IR）调度方法（Park 和 Kyung 1991）是一个启发式的调度算法，它有一个特点，那就是能够脱离局部最小值。每个箭头描述两个节点之间的一个优先级限制，如果这箭头是零延时的，那么这是一个内部迭代的优先级限制；如果箭头有一个或更多的延时，那么这就是一个相互之间迭代的优先级限制（Parhi 1999）。同时，内部的和交互的优先级限制规定了一个序数，依照这个序数，能够执行 SFG 中的节点。如果在非临界递归循环中的递归节点拥有循环灵活性（如将在之后阐述的一样），那么那个循环的节点能被移动到另一个没有违反内部和交互的优先级限制的时间分区中。对交互的优先级限制的探索产生了更好的调度并且通过使用循环的灵活性使得配置处理器的数量最小化。所有以上描述的算法都仅仅涉及内部迭代的优先级限制。Minnesota 架构综合（Minnesota Architecture Synthesis，MARS，Wang 和 Parhi 1994）开发了内部和交互迭代的优先级限制。

9.2 IRIS 行为综合工具

在第7章讲述了将 DSP 系统映射到 FPGA 中的设计工具的简洁性，讲述集中到基于 FPGA 架构的综合工具的设计上。但是，对每个工具仍然有一些限制，并且也有许多通过目前的系统不能有效处理的问题。其中一个例子就是通过流水线对系统行为和系统架构的作用产生的计算延时。即使 Handel-C、AccelChip（AccelFPGA 2002）、JHDL（Bellows 和 Hutchings 1998）和 MMAlpha（Derrien 和 Risset 2000）通过使用语法能够使得电路流水线化，并且 System Generator（Xilinx Inc. 2000）能通过改变 Xilinx 处理器模块的时延来使电路流水线化，但最后的时序问题仍有待设计者去解决。在高层次中，在自动实现电路层议题方面的工作量更小，比如数据时间格式、流水线重数及数值截断。此外，许多工具将架构的限制强加到设计者身上，这限制了对大部分架构选择的自由探索。特别的，设计者也许想利用硬件共享来研究解决之法，但是电路和相应的控制电路不能通过目前的工具自动产生。最后，这些工具已经不能依据具体的技术特征和比如流水线的低级设计技术来决定算法设计功能的架构。

本章中描述的这些，旨在为基于 FPGA 的 DSP 设计在架构的综合期间解决实现的问题。这些是基于目前的架构工具 IRIS（Trainor 等 1997），这个工具在 Queen's University Belfast 中得到了研发，这个工具原本是针对 VLSI 的。在 IRIS 架构的综合工具中相应的综合方法在接下来的章节中会介绍。

拥有在设计流程的每层实现最优化的综合工具是很重要的。在 20 世纪 90 年代中期，有人认为需要设计工具通过使用用户生成的处理装置来使得使用者能够从 SFG 表示式中快速地开发 VLSI 电路架构。由于这个原因，IRIS 架构的综合工具得到了开发（Trainor 等 1997）。它提供了一个通向低级的硅设计工具的有效通道。不像其他方法，在 IRIS 中的重点是为了生成基于那些装置的最优化架构，通过使用用户生成的处理装置使得算法的架构得到开发。这是基于使用由大公司提供的内部开发的处理器核的前提下的，并且因此也使得基于它们自己的处理核的解决方法的得到了开发。IRIS 的主要优势是它通过使用用户优先模块为设计者提供完全的自由来研究各种各样的架构。随着开发硅 IP 核，IRIS 的这个优势越来越重要，从而提高了这些使用者对优先模块的需要。这个工具也可以与传统 VHDL 综合工具相结合，并且已经被用于生产实际及现实的设计，因为它完全考虑了芯片层工程问题的影响。有一些设计问题，比如不同处理器的数据录入和录出的方式、数值的使用、流水线重数和数值截断的处理能改变整个系统的特点，特别是时序和时延（McGover 1993），并且因此改变系统的功能。提出了一个叫作模块化设计过程综合的方法（Modular Design Procedure，MDP，Trainor 1995）来为将技术映射到架构的高级算法和低级设计工具提供了一个桥梁。这成为了 IRIS的核心组件。

9.2.1　模块化设计过程

MDP 提供了一个将所有必要的实现标准合并到架构的推导过程中的方法。有两个处理器性能问题是要考虑的，即空时的数据格式和处理器时延的参数化（Trainor 1995）。

1. 在处理器输入和输出上的空时的数据格式

一个处理器的数据进入或离开的数据格式，或者“时序模型”可以被定义为数值的位数或比特数在时间上相对于其他数值的位置（McGovern 1993）。图 9-1所示为典型数据时序格式的一些例子。

流水线处理部件在细粒层上，也就是在处理器模块中的共同过程，导致了非平行的输出格式，因此一个处理这些非标准格式的工具是必要的。这可能与现代 FPGA 结构无关，在这个结构中，大多数加法器和乘法器以并行形式耗尽输入并且产生输出，如图 9-1a 所示。特殊处理器的详细结构，特别是内部的流水线寄存器的布置决定了在每个处理器输入和输出上的数据时序模型。在处理器时序模型上维护信息是有必要的，它可以决定需要将哪种额外的电路放置在连接的处理器之间，来使在第一个处理器的输出上的数据格式与在第二个处理器的输入上的预期格式之间进行转换。

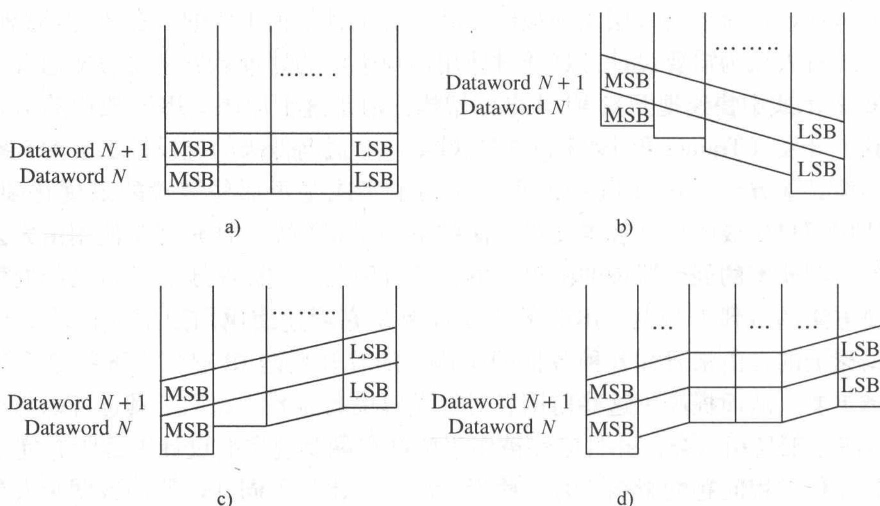


图 9-1 典型的数据时间格式

a）m 位并行：（0，0，…，0） b）m 位并行，lsb 优先：（$M-1$，$M-2$，…，1，0） c）m 位并行，lsb 优先：（0，1，…，$M-2$，$M-1$） d）m 位偏差，并行偏差（0，1，…，K，K，…，$K+1$，$K+2$，…）

2. 处理器时延的参数化

把在处理器的每个数据通道中的时延合并到在综合时使用的处理器模块中。主要原因是对单个处理器产生的结果而言是很必要的，时钟周期数已经很大地影响了贯穿整个架构的数据的时序。为了叙述 MDP 的运行，我们想到了一个简单的 Wallace 树乘法器架构处理器设计的例子，如图 9-2 所示。

在图 3-7 中给出的一个 Wallace 树乘法器包含了连接到一个快速加法器的 CSA。流水线化最显著的方法如图 9-2b 所示。为了使用连同 MDP 一起的 Wallace 树乘法器，必须决定适当的处理器性能值，也就是数据时序格式和参数化的数据通道时延，并且合并到特别的处理器模块中。和 Wallace 树一起的 9 比特操作数的部分乘积和的 MDP 例子如图 9-2c 所示。这个盒子代表了 CSA 处理器的树。在盒子中沿着一个特别的数据通道参数化的表达式代表那个数据通道的时延，同时数据时序模型在输入和输出的模式下清楚地显示了出来。时序模型是当数据的每一位通过乘法器时，通过考虑它的时序从而推导出来的。注意到一个处理器模式的结构已经抽象出了细节，仅剩下时延和数据时序模型的信息。除了字长参数及内部的流水线站，在处理器模型中的一些时延值也都依靠附加的记作 t 的截断来反映一个事实，那就是如果数值截断得到应用，则穿过那条数据通道的时延会增加。数据通道的时延被重定义为当输入的第一位进入到阵列时与当第一个可用位在输出出现时之间的时差。正如第 3 章中突出介绍的，在 DSP 应用中截断的建模是很重要的，虽然在处理器中用这种方法模拟截断也许很奇怪，但是这使得

系统层由截断决定的影响能够被考虑到系统的综合中，因为截断有时能提高输出的时延。例如，如果输出时序模型是一个偏差的格式并且需要数值截断，那么一定数量的 lsb 会被丢弃，并且在第一个可用位出现之前会需要考虑额外的时钟周期。在时延的表达式中，这些额外的周期在截断项中得到反映。这一项的值依靠输出数据时序模型，并且依靠它才能应用输出控制截断。IRIS 的关键特征是从 SFG 表示式中产生电路架构，它要求重定时过程的使用，这将在接下来阐述。

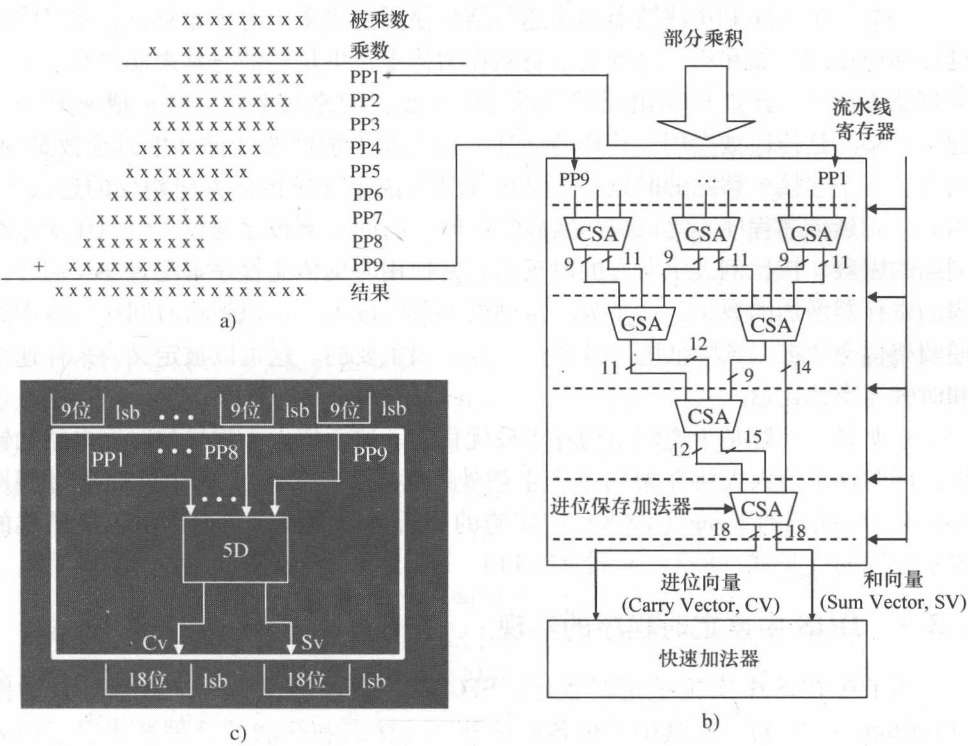


图 9-2　Wallace 树乘法器架构与 MDP 处理器模型的比较

9.3　IRIS 重定时

IRIS 起始点是综合算法的一个 SFG 表示法，在这个算法中，SFG 中的每个过程节点都被指定了精确流水线化处理器的一种模式。现在这些被定义了的处理器 MDP 模型能被嵌入到 SFG 表示式中。这个架构现在需要重新安排，也就是因为处理器时延中改变的重定时。这个重定时过程是基于早先在 8.4.2 节中描述的割集原理（Kung 1988，Leiserson 和 Saxe 1983）概括的重定时程序。与割集原理一样，IRIS 中的重定时过程包括两个步骤，也就是时延比例和时延传输。时延

比例使用在一个递归的架构中，也就是与反馈环一起的那种，这都是为了在反馈环四周引入足够的时延来保证时延传递的成功实施。在时延传递程序之后，需要在合适的 SFG 箭头上产生足够的时延，从而使得时延从图像箭头上移除并被合并到处理模块中，这是为了模仿数据通道的时延，又叫作将处理器模型嵌入到 SFG 中（Trainor 等 1997）。纯时延指的是剩在图表箭头处的任何额外的时延，并且它对纠正时序是必须的。这是 IRIS 中正确设计电路架构的基础。

当为一个如同 FIR 滤波器的非递归结构使用割集重定时程序时，会重复将时延传递应用到增加和移除时延上，直到在架构中产生足够的时延来使得其能嵌入处理器来代替。比如展示出反馈环的 IIR 滤波器的递归架构，会造成额外的问题。如果时延传递被应用于这样的循环，那么是不能够改变循环中时延数量的，因为从一个连接中移除的时延一定是传递到与这个连接传输方向相反的连接上。因此，时延传递程序不足以成功实施重定时。时间比例程序通过一个叫作流水线周期的因数，按比例表示所有的时延值，并使用时延传递程序重配置在反馈环四周的寄存器增加的数量，这是为了帮助处理器的嵌入。认识到将时间比例应用到递归架构会依据因数降低电路效率这一点是很重要的，这可以被定义为采样速率和时钟速率的比值。

很明显，问题是为递归结构寻求最优化值。这个值太小将导致时序电路的错误，同时这个值太大将在架构中产生额外不必要的寄存器。这个问题通过利用 8.4.3 节中强调的 Kung（1988）中实施的分析得以解决，它确定出了最糟糕的流水线周期（见式（8-3）和式（8-4））。

9.3.1 IRIS 中重定时程序的实现

为了在 IRIS 中实施重定时程序，SFG 原理图被建模为一个双倍加权的图（Christofides 1975），在其中各种各样的外部连接器和算法的处理器代表了图表的节点和这些节点之间代表图表箭头的连接。对每个箭头，两个加权可以定义为 SFGW 和 PW. SFGW，以前在论文中（Yi 等 2005）叫作 SFG Weight，它指的是在一个特别的箭头上迟延元件的数量。并且 PW（以前的 ProWeight）指的是被要求在那个箭头上为了处理器嵌入而存在的时延元件的数量。最大的允许的采样周期是式（9-1）所定义的。

为了使过程自动化，IRIS 使用了来自图论的 Floy - Warshall 算法（Parhi 1999），它决定了在双权重图中最快的循环。这个算法解决了寻找一个满足式（9-2）的循环 ϕ 的问题，在这个等式中，在加权图中是一个箭头。由于算法已经被转换为了一个最小化的问题，因此目前 α 的值是 $Z(\phi)$ 最小值的倒数（Trainor 等 1997）。

$$\alpha = \max\left[\frac{\sum_{e\in\phi}\mathrm{ProcWeight}(e)}{\sum_{e\in\phi}\mathrm{SFGWeight}(e)}\right] \tag{9-1}$$

$$Z(\phi) = \max\left[\frac{\sum_{e\in\phi}\mathrm{ProcWeight}(e)}{\sum_{e\in\phi}\mathrm{SFGWeight}(e)}\right] \tag{9-2}$$

若流水线周期已经被决定，则所有的 SFG 中时延元件都通过这个值按比例放大，它等价于将整个结构的速率调整到它最慢的循环。

在时延比例后，需要应用重定时程序，这是为了补偿一般的 SFG 处理器和实际模型之间不同的性能特点。在 IRIS 中作为一个线性编程问题已经得到解决的重定时程序使得使用的时延总数最小化。一个 SFG 的重定时给节点 G 赋了一个整数值。$r(v)$ 值是是从节点 v 的每一个进来的箭头中提取的和被推向输出箭头的时延的数量。如果时延从输入移动到输出，那么 $r(v)$ 值是正数，否则是负数。式（9-3）表示线性编程问题的目标函数，其中 $\beta(e)$ 表示箭头的宽度，符号 $e(v\rightarrow?)$ 和符号 $e(?\rightarrow v)$ 表示在节点上箭头各自的起点和终点。线性约束的方程涉及构造式（9-4）的形式的表达式，在这个表达式中，箭头离开节点 u 并进入节点 v。在 IRIS 中的重定时功能利用了改进的单形技术（Gnizio 1985），这项技术广泛地应用于自动化的线性编程解算器。

$$\min \sum_{v\in V} r(v)\left(\sum_{e(v\rightarrow?)}\beta(e) - \sum_{e(?\rightarrow v)}\beta(e)\right) \tag{9-3}$$

$$r(u) - r(v) \geqslant \mathrm{ProcWeight}(e) - \mathrm{SFGWeight}(e) \tag{9-4}$$

为了说明在 IRIS 中重定时程序的过程，使用了早先在式（8-3）中给出的 2 阶 IIR 滤波器。这个例子在 8.4.3 节中得到了实现，但是在这里再一次给出是为了说明 IRIS 是如何运行的。图 9-3a 所示为一个 SFG，其中 MAC 处理器被假定拥有零时延。

高速的应用通常要求流水线的设计，因此所有的电路节点都通过使用流水线式的 MAC 处理器实现，在这个处理器中，加法器和乘法器都有一个流水线站。在图 9-3b 中注意到在结构中每个连接的“宽度”等于沿着那个连接传输的数据的字长。需要使用在图 9-3b 中显示的节点的参数值来计算滤波器设计具体的时延值和数据时序模型。

在这个例子中，MAC 处理器架构和相应的 MDP 模型如图 9-3b 所示。一个输出的时延等于所有从输入到这个输出的数据通道的最大时延。比如，在流水线式的 MAC 处理器中从每个输入到输出有三个数据通道 S_o。输出 S_o 的时延是 2，它等于从输入 P_i，Q_i 到输出 S_o 的时延。有着最大时延的通道被用于在 0 时钟周期时定义输入数据时序的格式。随着在这个模型中输入或输出的所有位同时进入

或离开处理器，数据时序格式是［0，0，0，…，0］，称其为［0］。所有的其他输入和输出数据时序格式需要参考这个通道。比如，信号的时延 S_o 等于2并且是所有输出时延的最大值。因为信号 P_i 和 Q_i 数据时序格式等于如图9-3b所示的［0，0，0，0，0，0，0，0］，并且从 S_i 到 Q_i 的时延是1，信号 S_i 应该在信号 P_i 和 Q_i 之后进入处理器的一个时钟中，因此在处理器MDP模型中，数据时间格式是［1，1，1，1，1，1，1，1］，称其为［+1］。图9-3a的SFG所示为两个在图9-3a中强调的循环。通过使用在图9-3b中所示的MDP模型，评估式（9-1），给出如式（9-5）的结论。

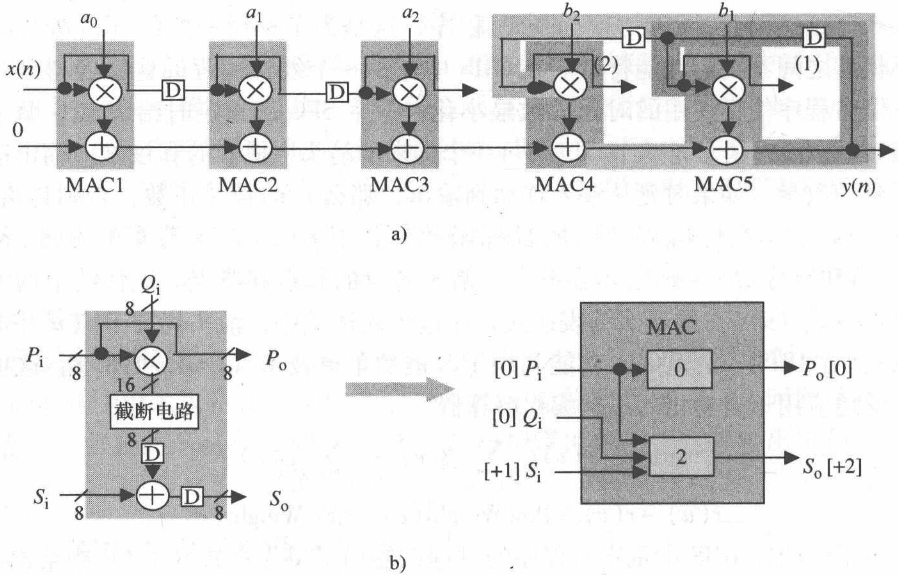


图9-3　第二级IIR滤波器

a）使用零延迟MAC的IIR滤波器的实现　b）流水线式处理器和MDP模型

注意到循环2的循环界限应该等于（1+1+1）/（1+1）=1-5，它不同于 α_2。原因是输入上的数据时序格式也许是在一个不同的时间上，这个时间在时延比例和IRIS时延传递重定时程序中没有被考虑到。考虑到循环2中输出 S_o（MAC4）和输入 S_i（MAC5）之间的连接，S_i 的数据时序格式是［+1］，它意味着输入数据进入到MAC5中比其他的输入（P_i 和 Q_i）要晚一个时钟周期。当两个时延都进入到处理器MAC5中来合并流水线处理器时，应该将一个额外的时延添加到输入 S_i（MAC5）的箭头中。这个额外的时延能被用于模拟MAC4通过流水线造成的输出时延。第二级IIR滤波器的流水线周期是2，并且因此所有滤波器SFG中的时延元件必须在重定时发生之前按因数2的比例表示。在时延比例和重定时之后，修改后的SFG如图9-4所示。

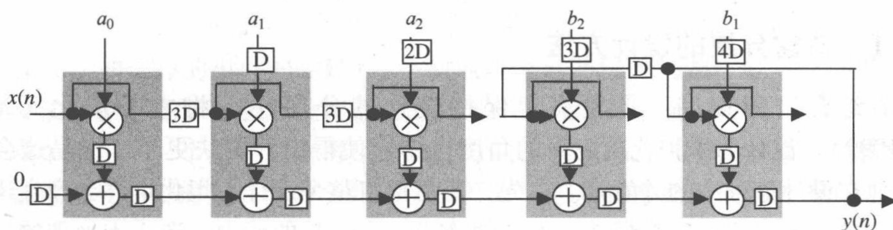


图 9-4　使用流水线的 MAC 的第二级 IIR 滤波器的实现

$$\alpha_1 = \left[\frac{\sum_{e\in\phi}\mathrm{PW}(e)}{\sum_{e\in\phi}\mathrm{SFGW}(e)}\right] = \frac{2}{1} = 2\{\text{循环 }1\}$$

$$\alpha_2 = \left[\frac{\sum_{e\in\phi}\mathrm{PW}(e)}{\sum_{e\in\phi}\mathrm{SFGW}(e)}\right] = \frac{2+2}{1+1} = 2\{\text{循环 }2\}$$

$$\alpha = \max(\alpha_1, \alpha_2) = 2 \tag{9-5}$$

此图展示了输入需要通过输入上时延数量的指示进入设计中的时间点。实际上，这些时延不需要在电路中得到实现，但是可以表示在正确的运行中需要的控制。

9.4　分层的设计方法

在以前章节中的描述已经表明了在第 8 章中概述的重定时技术是如何在 IRIS 中自动化的。工具流的输出是一个最优化的电路架构，它能以 VHDL 编码并且通过使用 FPGA 布线和路由设计工具得到实现。这个例子描述了 DSP 的功能，在这个功能中，这个设计像一个平面 SFG 的表示式得以实现。逐渐的，DSP 硬件流需要应对分层，其中涉及了复杂的组件。主要的，以一种分层的风格构建设计，其中会产生子组件然后被用于构建更大的系统。当以提议的方法应用重定时时，会带来具体的问题，比如，这些子组件不能像黑盒组件一样被简单地对待，因为它们包含了内部定时。

本节描述了分层是怎么合并到设计流程中的，并且介绍了白盒分层管理的概念，它使得之前产生的核的内部架构有一些改变。这个挑战通过使用一个波数字椭圆（Wave Digital Elliptic，WDE）滤波器例子（Lawson 和 Mirzai　1990）得到了叙述，这个例子通过使用原始的 IRIS 工具能被综合。因此，使用原始 IRIS 工具阐述分层的电路的定时问题是一个好例子。然后提出了白盒方法并且应用于综合分层的结构。也提出了对 MDP 的一个修正来处理分层的自动的综合问题。

9.4.1 白盒分层的设计方法

在综合过程期间，必须决定保持设计的分层还是将之平坦化（Xinx Inc. 1999）。设计的平坦化从逻辑的角度也许会使得设计更快更小，但是这会产生其他在设计流程中的并发症。首先，盲目地将整个设计平坦化也许会产生逻辑的一个单一的模块，这个模块会有足够淹没综合工具的能力，造成难处理的运行次数或产生一个次最优的网表。此外，也许还会产生一个高度杂乱的网表，从而不能维持各种子系统的规律。随着更高密度的 FPGA 的引入，分层设计的优势带来的收益使得任何缺点都可忽略不计。

一个分层的方法相比于一个平坦的方法的缺点是设计映射不能最佳地跨过分层边界。当这使得设备无效率并且降低设计性能时，分层的方法却能构建出有效率的设计分区，将单个模块的选择和增加的设计变化混合在了一起，也能使得设计流程的管理更有效，并且通过开发再用，减少了设计和调试时间。为了受益于分层的方法，需要有效率的策略来划分设计，最优化的分层及调整分层的综合过程。

分层的设计流程的意义与 IP 核再用的观念是一样的（Keating 和 Bricaud 1998），在其中，由于一系列系统参数，能提前产生出设计规划，从而使得这些参数能得到广泛的应用（McCanny 等 1997）。这在第 10 章和第 12 章中将得到详细的讲述。因此，IP 核的使用本质上是一个分层的方法。

有两种不同的分层的综合方法已经由 Bringmann 和 Rosenstiel 定义，并且都是基于：

1）黑盒的再用，在里面以前综合的系统作为不能进入内部结构的组件；

2）白盒的再用，在里面以前综合的系统作为由于流水线或重定时而改变内部结构可能性的组件。

这两种方法都是与 IP 核设计方法相称的，在这种方法中，系统设计者也许有有限的知识，但却能控制复杂组件。然后就有可能开发产生数据通道时延和输入定时的模型，并以一个与在 IRIS 中提出的类似的分层方法来执行综合。正如通过使用 WDE 滤波器例子（Parhi 1999）将演示的，这会事与愿违并且会产生非常无效率的解决之法。

白盒分层的管理在 Synplify Pro 综合工具中得到使用。在综合过程期间，那些工具会解散尽可能多的分层使得逻辑能有效并且最优地跨过分层间的边界并且维持着快速运行速度（Drost 1999）。然后 Synplify Pro 尽可能紧密地对原型进行重建分层，这保留了跨越分层边界的最优化。这产生了一个最后的网表，这个网表拥有与原始源代码同样的分层，这保证了分层的寄存器名始终如一，并且逻辑主要模块依然集中在一起。这个结合架构的具体映射来处理分层边界的方法，产

生了一个高效且有效的最优化引擎，但是，设计者需要根据 Synplify Pro 中组件的选择（比如选择的组件的时延），手动地研究电路层的问题。电路层问题的手动解决法减少了在算法层上设计空间的开发，并且增加了上市时间。

本节展示了一个基于白盒再用的设计方法，它使得在没有影响整个架构的情况下能够改变内部时延。随着分级方法的开发，需要将电路层问题上的自动化的综合方法添加到分层设计上。在接下来的章节中，会介绍在分层电路中从以前综合的子系统中提取 MDP 模式。

9.4.2 从以前的综合架构中提取处理器模型的自动化实现

分层的定时问题通过使用一个在从 Parhi（1999）中获取的图 9-5a 中简单的 5 阶 WDE 滤波器电路来演示。如第 2 章简洁描述的，WDF 滤波器有着极好的稳定性（甚至在非线性操作条件下产生溢出和凑整的影响）、低系数的字长要求、固有的好的动态范围等（Gazsi 1985）。它们非常有吸引力，这都归因于它们对系数量化的低敏感度（Lawson 和 Mirzai 1990）。5 阶 WDE 滤波器是根据模型 A、B、C 和 D 子系统构思的，如图 9-5b ~ e 所示，并且包含了乘法运算和加法操作。

提出了许多步骤来以分层的方式处理这个设计。首先，用公式表示基本算法操作的处理器模型。为了取得一个高性能 FPGA 实现，假定每个乘法器和加法器都在它们的输出上拥有一个流水线寄存器，即使流水线的各层能在 IRIS 中模拟。这与 FPGA 的实现相称，在这个 FPGA 实现中，在组件层使用流水线可能是最好的选择。接下来，使用基础的算法处理器来综合模块子系统的 SFG。然后使用综合的模型架构来衍生等价的 IRIS 处理器。最后，使用模型 A、B、C 和 D 子系统的处理器模型来综合 5 阶 WDE 滤波器的 SFG。

主要的困难是一些模型，也就是 A 和 D，也许包括能影响决定流水线周期过程的寄存器。第一，模型 D 子系统中的时延在高层电路中造成了错误的流水线周期。第二，凭借原始 IRIS 能自动从先前综合的架构中提取出处理器模型的技术还没有得以实现，因此，需要一个新技术。最后，这有一些实例，在这些实例中，分层的电路不能被综合，即使能够正确地计算流水线周期，这是因为由IRIS（Trainor 1995，Trainor 等 1997）使用的线性编程不能实现重定时。供代替的选择是使分层的 SFG 层平坦化直到电路能被综合。后面将会详细地讨论这些方法。注意到一点是很重要的，那就是当研究分层的设计时，这些定时问题比起 IRIS 是能被应用于更多种工具上的。

在分层的 SFG 中，会提前决定 MDP 模型的每个处理器的流水线周期。因此，分层的 SFG 流水线周期（Parhi 1999）的计算成为一个在 IRIS 中较复杂的任务。为了计算整个电路的流水线周期，有必要假定流水线周期等于 1 然后使得

IRIS能将它改变成所有模型的最大流水线周期。当 IRIS 综合和重定时程序被应用于如图 9-5b ~ e 所示的模型 A ~ D 的模型层子系统时，7 个时钟周期的最大流水线周期得到计算，它是下面描述的模块 D（α_D =7）的流水线周期。因此，整个电路流水线周期改变到了 7。

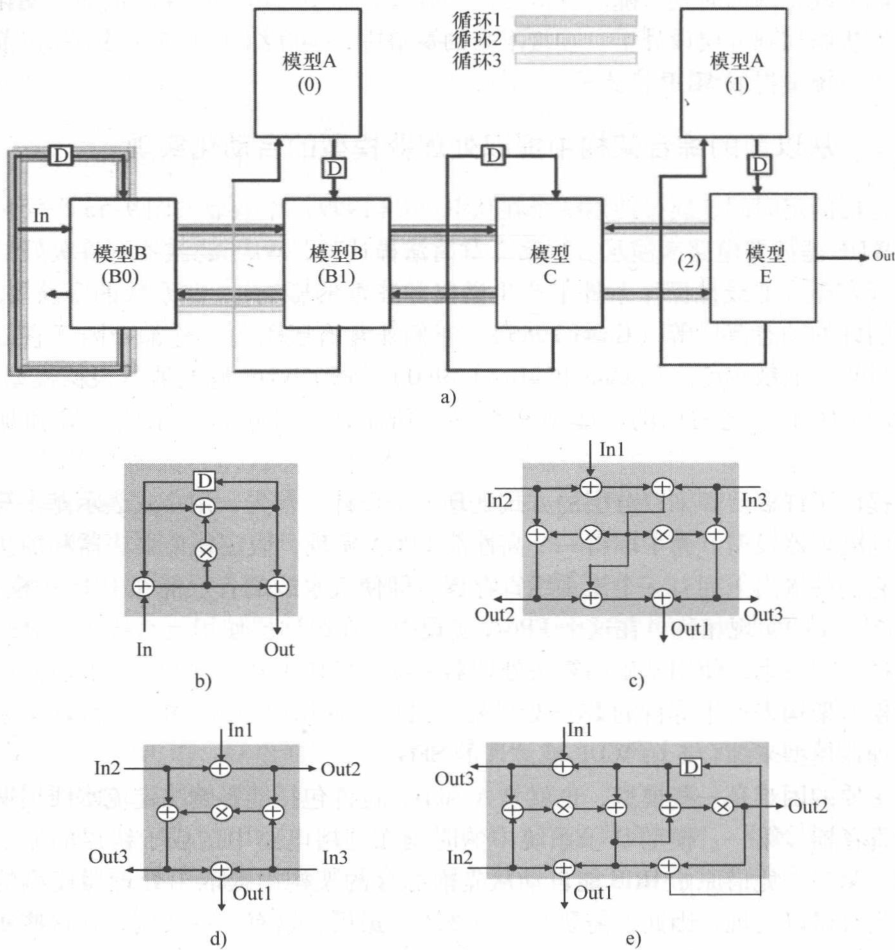


图 9-5 5 阶 WDE 滤波器的 SFG（TA 和 TM 的流水线是 0）。摘自 *Hierarchical Synthesis of Complex DSP Functions Using IRIS* by Y. Yi & R. Woods, IEEE Trans on Computer Aided Design, Vol. 25, No. 5, © 2006 IEEE

a）完全的 SFG b）模型 A c）模型 B d）模型 C e）模型 D

模型 A：循环限制 =3；

模型 B：无循环；

模型 C：无循环；

模型D：循环限制=7。

IRIS确定当子系统架构被转变成一个处理器模型时，子系统模型中时延的数量会尽可能少。因为这些时延不能加入到在整个5阶WDE滤波器的综合期间发起的重定时进程中，这提供了一个更好的解决之法。因此，任何对改变模型A中的输入/输出数据流必需的重定时锁存器会出现在不同的处理器实体之间。这使得IRIS在5阶WDE滤波器中的各种子系统内部和之间使用重定时锁存器的总数的最小化更易实现。

对模型A子系统的综合架构如图9-6a所示。输入和输出的重定时值能通过使用一个改进的单一线性编程来获得（Vajda 1981）。一个输入的重定时值指的是从子系统外面移动到子系统里面的时延的数量。比如，输入的重定时值In为4，这意味着4个时延被移动到了子系统中。输出的重定时值Out为0，意味着没有时延从子系统移动到外面。

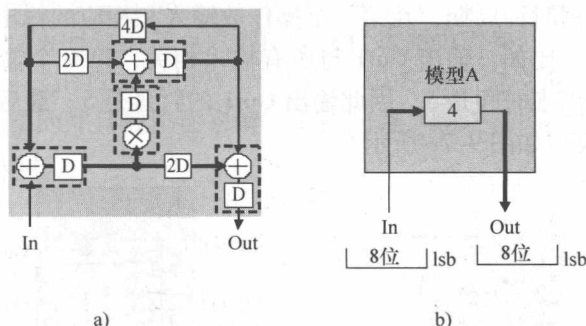


图9-6 模型A子系统（TM=1及TA=1）。摘自*Hierarchical Synthesis of Complex DSP Functions Using IRIS* by Y. Yi & R. Woods，IEEE Trans on Computer Aided Design，Vol. 25，No. 5，© 2006 IEEE

a）综合架构 b）MDP模型

为了产生如图9-6a所示的重定时的模型A架构的MDP模型，首先研究如图9-5b所示的模型A子系统的原始SFG的算法表示式，在里面乘法器和加法器的时延是0。输入和输出分别同时抵达和离开子系统，因此输入和输出的数据时序格式是0并且输出的时延也是0。在乘法器和加法器时延改变为1之后，IRIS重定时子程序被用于处理由乘法器和加法器的时延造成的时序问题。输入In的重定时值就是当一个处理器的时延改变时，需要的时延的数量，并且会被添加到输入。输出Out的数据时序格式将会被改变到4，因为输出Out在4个时钟周期之后出现。综合的架构会在将4个时延移动到模型A中并且执行重定时之后产生。模型A子系统实现的MDP表示式如图9-6b所示。

有一个输入和一个输出的模型A子系统是一个简单的子系统，并且因此将

使用模型 B 子系统时延多点输入和输出的提取方法。模型 B 子系统中循环的流水线周期均不为 7（等于系统流水线周期）。在原始模型 B 中，同时出现的所有输入输出和它们的数据时序格式及输出时延都是 0。重定时之后，模型 B 子系统的各个输入和输出也许会在不同时钟周期中出现，如图 9-7a 所示，其中，输入的重定时值，也就是 In1 和输出都如括号中所示，比如（+5）。

IRIS 将子系统中所有输入的最大的重定时值定为 β，并且将［$+\beta$］时延添加到所有输入中。所有输出的数据时序格式都将被改变为 β，因为 β 时延被嵌入到了输入中。在 IRIS 重定时之后，输入箭头中时延的数量将减小那个输入的重定时值，并且等于重定时值的时延数量，将被添加到输出上。对于模型 B，β 值为 5，并且这些时延被添加到所有输入中。重定时之后，在输入 In3 箭头上的时延的数量被改变到 1 并且等于 β 值减去输入 In3 的重定时值。输出 Out2 箭头的时延数量是 Out2 的重定时值并且等于 4。整个程序如图 9-7b 所示。一个输出的时延被定义为根据时钟周期，在第一个操作数输入和相应的输出的第一个操作数出现之间的时差。比如，输出 Out1 与所有输入相关，第一个输入在 0 时钟周期时输入，输出的出现时间是 5，因此输出 Out1 的时延是 5。最后将被用在下一层综合中的 MDP 模型如图 9-7c 所示。

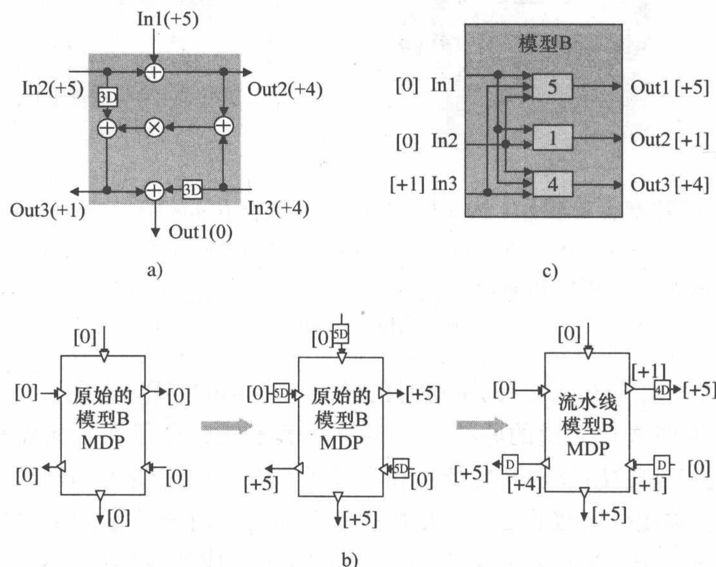


图 9-7　模型 B 子系统（TM = 1 并且 TA = 1）。摘自 *Hierarchical Synthesis of Complex DSP Functions Using IRIS* by Y. Yi & R. Woods, IEEE Trans on Computer Aided Design, Vol. 25, No. 5, © 2006 IEEE

a）综合架构　b）模型 MDP 推导进程　c）MDP 模型

9.4.3 IRIS 中分层的电路实现

对所有不同的处理子系统而言，一旦获得了在先前章节中描述的那种类型的处理器模型，那么合适的模型会被嵌入到更高层 SFG 的处理节点中。如前面所述，在更高层的电路中需要架构的重新安排和重定时，那是因为在更低层处理器时延中使用的流水线处理器的改变。

分层架构的重定时在两个阶段中实行：时延比例和处理器嵌入（Kung 1988）。但是，先前使用的双加权图表示式不足以解决分层电路中的定时问题，是因为输入可以出现在不同的时间点。例如，在模型 B 子系统中输出 Out3 的时延是5，因此在输出 Out3 箭头中需要5 个时延来嵌入到处理器中。额外的一个时延会在处理器嵌入后被添加到输入 In3 箭头中。原始的 IRIS 工具不能代表加权图中额外的一个时延，也因此不能有效地解决问题。为了处理这个问题，引入了三加权图，其中各种外部连接器、算法处理器和允许信号通道分支的电线接头代表了图中的节点，并且这些节点之间的连接代表图中箭头。对于每个箭头，三加权得到如下定义。当 SFGW 和 PW 因为原始 IRIS 得到定义时，PWI（或严格的 ProWeightIn）是新的并且被添加来解决上述问题，随之，PW 被改变为 PWO，也就是 ProcWeightOut。

SFGW——一个特殊箭头上时延的数量；

PWO——处理器嵌入所需要的产生时延的数量；

PWI——为了处理器嵌入，添加到箭头上时延的数量。

箭头的 PWO 值与起始处理器的输出有关并且是输出的时延，这个时延是输出引脚的数据时序格式表达式的最小值（这是为了迎合这里没有讨论的情况，在这种情况下数据是位偏移且不是位并行的）。箭头的 PWI 值与最终处理器的输入信号有关，并且等于输入引脚的数据时序格式表达式的最小值。箭头的 PWO 和 PWI 值与连接器和电线接头有关并且等于0。流水线周期由于分层的电路更改为式（9-6）且线性编程问题的目标函数和限制如式（9-7）所示。新的 PWI 参数因为分层设计被添加到正确的式（9-1）和式（9-4）中。

$$\alpha = \max\left[\frac{\sum_{e\in\phi}\mathrm{PWO}(e) - \sum_{e\in\phi}\mathrm{PWI}(e)}{\sum_{e\in\phi}\mathrm{SFGW}(e)}\right] \tag{9-6}$$

$$\min \sum_{v\in V} r(v)\Big(\sum_{e(v\to ?)}\beta(e) - \sum_{e(?\to v)}\beta(e)\Big)$$
$$r(u) - r(v) \geqslant \mathrm{PWO}(e) - \mathrm{SFGW}(e) - \mathrm{PWI}(e) \tag{9-7}$$

随着低级电路图中子系统的算法的时延能决定更高级电路的流水线周期，通过使用式（9-6）计算流水线周期，分层 SFG 的（α）成为 IRIS 中一个复杂的任务。在子系统的 MDP 模型产生之后，IRIS 将计算整个5 阶 WDE 滤波器流水线周期，将在以下章节中介绍。

9.4.4 分层电路中流水线周期的计算

在一个 5 阶 WDE 滤波器的例子中，模型 A 和 D 包含了一些寄存器，这些寄存器会影响由 IRIS 重定时子程序决定的流水线周期。从之前的章节中，模块层子系统的流水线周期为 7，它比最初值的 1 更好。因此，整个系统流水线周期改变到 7。模型 A 和 B 子系统的综合 MDP 模型如图 9-6b 和 9-7c 所示。一样的自动处理器模型提取物被应用到模型 C 和 D 中，并且综合电路和图形化的表达式如图 9-8 和图 9-9 所示。

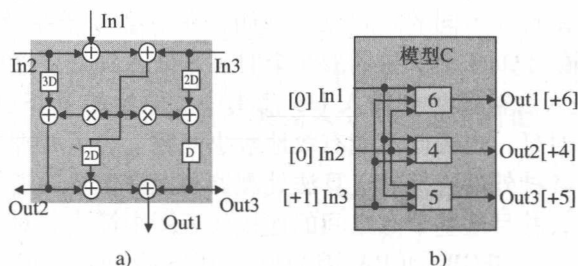


图 9-8 模型 C 子系统（TM = 1 和 TA = 1）
a）综合架构 b）MDP 模型

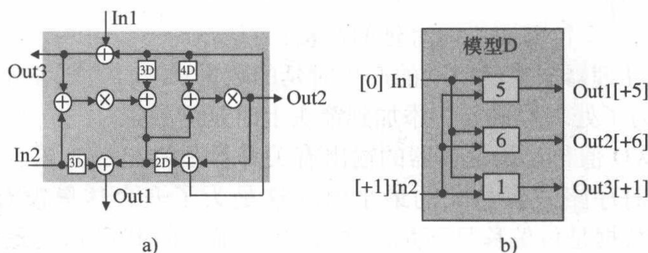


图 9-9 模型 D 子系统（TM = 1 和 TA = 1）
a）综合架构 b）MDP 模型

一旦获得了所有的处理器模型，合适的模型就会被嵌入到更高层 SFG 的处理节点中，正如之前所阐述的，在更高层的电路中需要架构的重定时，那是因为更低级的处理器时延中的变化。

由于在原始 SFG 子系统中的时延在更高级的分层综合中不能在式（9-6）中用作 SFGW，因此通过使用式（9-6），分层架构的流水线周期会在许多情况下产生错误。例如，5 阶 WDE 滤波器（见图 9-5a）展示了以下两个递归循环。通过使用 PWO/PWI 和 SFGW 值，带入式（9-6）后产生了以下结论。

循环 1：B（0）→B（1）→C→B（1）→B（0）→D

$$\alpha_1 = \frac{\sum_{e \in \phi} \mathrm{ProcWeightOut}(e) - \sum_{e \in \phi} \mathrm{ProcWeightIn}(e)}{\sum_{e \in \phi} \mathrm{SFGWeight}(e)}$$

$$= \frac{1+1+4+4+5-0-0-1-1-0}{1} = 13$$

循环 2：模型 B（0）→模型 B（1）→模型 C→模型 D→模型 C→模型 B（1）→模型 B（0）→D

$$\alpha_2 = \frac{\sum_{e \in \phi} \mathrm{ProcWeightOut}(e) - \sum_{e \in \phi} \mathrm{ProcWeightIn}(e)}{\sum_{e \in \phi} \mathrm{SFGWeight}(e)}$$

$$= \frac{1+1+5+1+4+4+5-0-0-1-1-1-1-0}{1} = 17$$

关键循环是循环 2，它的流水线周期是 17，但是循环 2 数据通道包括了模型 D 子系统中的一个算法时延，并且它的流水线周期应该是 12，也就是[23/(1+1)]。此外，当式（9-6）被用于计算如图 9-5 所示的循环 3 时，流水线周期等于无穷。这是因为模型 B 子系统的输出 Out2 仅与输入 In1 和 In2 有关。模型 B（0）和模型 B（1）之间的连接不能形成一个循环。为了解决这两个问题，除了数据时序格式和时延外，需要添加一些其他参数到输出上。

引入一个输出的相关输入矢量并且描述了由一个数据通道连接到那个输出的输入。比如，在模型 B 中的输出 Out2 的相关输入向量是［In1，In2］。模型 A ~ D 的相关输入向量见表 9-1。如果在一个子系统中从输入 u 到 v 有一条通道，那么会计算出称作数据通道时延对的信息（时延和函数时延）。函数时延不能提高时延，但是会产生校正函数并能用作 SFGW。其中的时延指的是流水线时延并且用作 PWO。如果在 u 和 v 之间有多重数据通道，那么函数时延的值将不同，并且 I-RIS 需要所有的信息对来计算流水线周期。如果数据通道时延中的函数时延是一样的，那么将选择最大的时延值来计算流水线周期。如果子系统不包括函数时延，那么就不需要数据通道时延对，因为能使用 MDP 模型三倍加权来计算流水线周期。如果子系统包括了函数时延，将同时使用 MDP 模型和数据通道对来计算流水线周期。MDP 模型决定 PWI 并且数据通道时延对决定 PWO 和 SF-GW 值。包括函数时延的模型 A 和 D 的数据时延对见表 9-2。MDP 现在包括了 4 个参数，分别是数据时序格式、数据通道时延、相关输入向量和数据通道时延对。

表 9-1　模型 A ~ 模型 D 输出关系

输出	模型 A	模型 B	模型 C	模型 D
Out1	[In 1]	[In1, In2, In3]	[In1, In2, In3]	[In1, In2]
Out2		[In1, In2]	[In1, In2, In3]	[In1, In2]
Out3		[In1, In2, In3]	[In1, In2, In3]	[In1, In2]

表 9-2 模型 A 和模型 D 的数据通道时延对比

	模型 A	模型 D	
	In 1	In 1	In 2
Out 1	(4, 0) (5, 1)	(5, 0) (12, 1)	(4, 0) (11, 1)
Out 2		(6, 0) (13, 1)	(5, 0) (12, 1)
Out 3		(1, 0) (8, 1)	(7, 1)

通过应用这个方法，能够计算更高级电路的流水线周期。IRIS 检测 SFG 中的循环并决定时延比例是否必要。在高级的 5 阶 WDE 滤波器中有 25 个循环。关键循环的计算如下所示，并且此电路的流水线周期是 13。

循环 1：模型 B（0）→模型 B（1）→模型 C→模型 B（1）→模型 B（0）→D

$$\alpha_1 = \frac{\sum_{e\in\phi} \mathrm{PWO}(e) - \sum_{e\in\phi} \mathrm{PWI}(e)}{\sum_{e\in\phi} \mathrm{SFGW}(e)} = \frac{1+1+4+4+5-0-0-1-1-0}{1} = 13$$

循环 2：模型 B（1）→模型 C→模型 B（1）→模型 A→D

$$\alpha_1 = \frac{\sum_{e\in\phi} \mathrm{PWO}(e) - \sum_{e\in\phi} \mathrm{PWI}(e)}{\sum_{e\in\phi} \mathrm{SFGW}(e)} = \frac{1+4+5-0-1-0-0}{1} = 13$$

9.4.5 分层电路中的重定时技术

在时延比例之后，使用线性编程程序来实施处理器的嵌入。在一些场合，甚至当能够正确计算流水线周期时，分层电路仍不能被综合，因为线性编程没有解决之法。IRIS 的重定时程序将从最高级实行综合并产生电路。否则，会使有冲突的子系统平坦化然后再综合电路。如果在这一层电路综合的结果不存在，则IRIS工具反复的将冲突的子系统平坦化并且重定时直到电路能够被综合。目前，IRIS 工具不能自动地做这件事。

线性编程没有解决之法，因为一些限制是有矛盾的。比如，在式（9-7）中，从模型 B（0）的输出 Out2 到模型 B（1）的输入 In2 的箭头的限制是 $r(B0)-r(B1)\geqslant 1$，并且从模型 B（1）的输出 Out3 到模型 B（0）的输入 In3 的箭头限制是 $r(B1)-r(B0)\geqslant 3$。这两个限制是冲突的，因此线性编程没有解决之法。

在这种情况下，IRIS 重定时程序会让模型 B（0）、B（1）、C 和 D 平坦化、执行重定时并且将平坦化的电路转换回分层电路。5 阶 WDE 滤波器的部分平坦化的 SFG 的三倍加权图如图 9-10 所示，在其中平坦化子系统的输入和输出是为了 PWO 和 PWI 并被用作图形节点。重定时之后，对 5 阶 WDE 滤波器的综合架构如图 9-11 所示。它能被转换回分层的表示式。比如，模型 B（0）子系统在重定时过程中被平坦化且模型 B（0）的输入和输出被做了记号，并且与三倍加权图形的相应节点相关联，这都是为了在重定时后被转换回模型 B（0）子系统。为了检测这个技术的正确性，5 阶 WDE 滤波器的整个平坦化的电路通过使用 IRIS 工具得到增强并综合。重定时的 SFG 如图 9-11 所示。

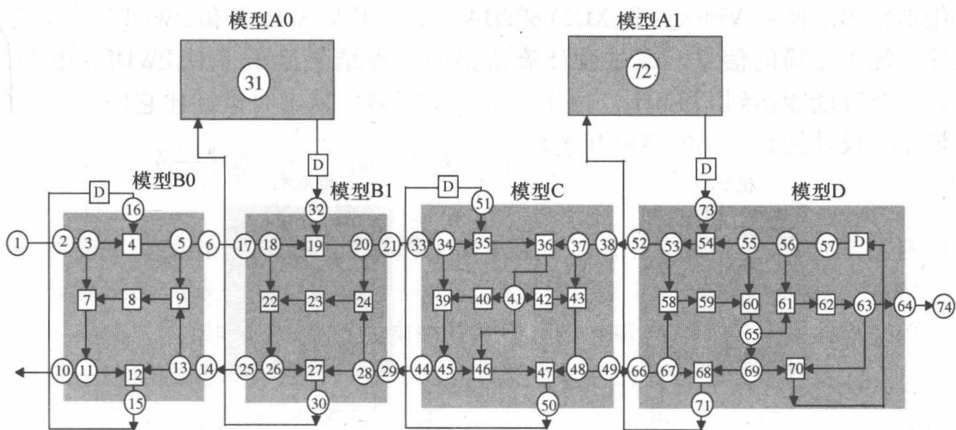


图 9-10　5 阶 WDE 滤波器的部分平坦化 SFG 的三倍加权图

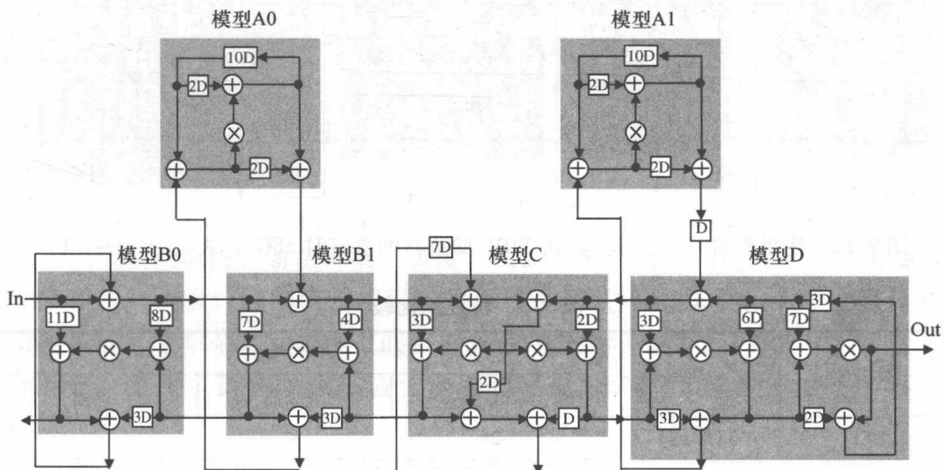


图 9-11　TA = 1 和 TM = 1 的 5 阶 WDE 滤波器流水线层的重定时 SFG

在综合期间，IRIS 会产生 5 阶 WDE 滤波器电路结构化的 VHDL 描述。定义

了每个处理器类型（乘法器和加法器）和时延模块并且规范了它们的互连。会产生模块（A0，A1，B0，B1，C 和 D）和滤波器架构的独立代码块。这使得整个分层的 VHDL 描述由于逻辑综合被存入 Synplify Pro，并且因此经 Xilinx Design Manager 被存入物理的实现中。

5 阶 WDE 滤波器在关键路径和使用硬件方面的比较见表 9-3。图 9-5 所示的 WDEF 是原始滤波器，图 9-11 所示的 PWDEF 是最优化的设计。图 9-12 所示的电路是由原始 IRIS 工具产生的原始的流水线式 WDE 滤波器（Original Pipelined WDE Filter，OPWDEF）。PWDEF 和 OPWDEF 电路的关键路径减少了93%，超过了 WDEF 电路。PWDEF 相较于 OPWDEF 在时延上下降了 27%。表 9-4 和表 9-5 详细地给出了使用 Virtex－Ⅱ XC2V80fg144－5 芯片后 WDEF 和 PWDEF 系统的 8 位输入输出布局的信息。通过设计流获得的实现结果显示了对 PWDEF 电路而言，系统性能能达到 173MHz。由于流水线寄存器，重定时设计比它们相对应的无流水线设计拥有更小的路由互连时延。

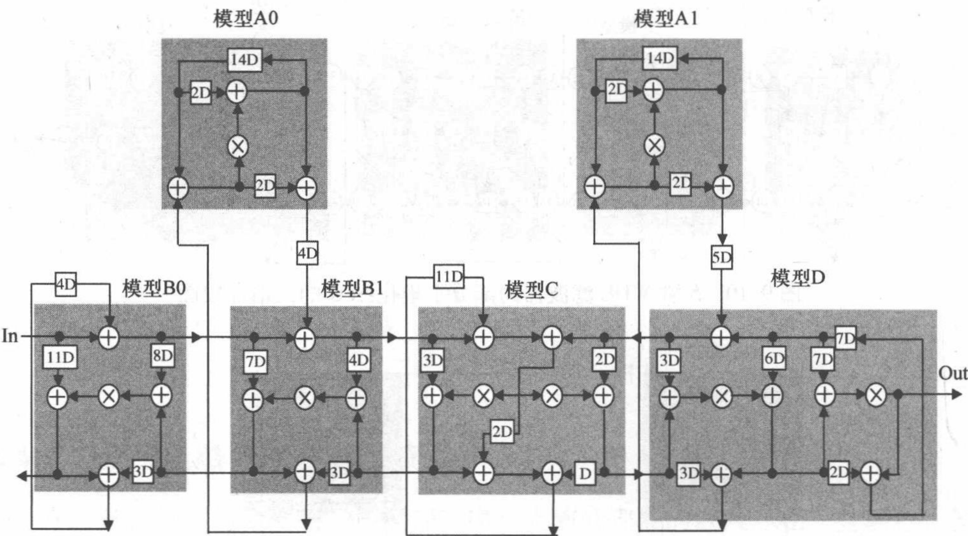


图 9-12 使用原始 IRIS 工具，5 阶 WDE 滤波器的重定时 SFG（TA = 1 且 TM = 1）

表 9-3 特征的比较

架构	关键路径	硬件元件		
		加法器	乘法器	时延
WDEF	3TM + 11TA	26	8	7
PWDEF	TM	26	8	104
OPWDEF	TM	26	8	132

表 9-4　Virtex－Ⅱ对于采样率的布局后的结果

电路名称	时钟（MHz）	关键路径（ns）		
		逻辑	路由	总的
WDEF	25.6	24.2	14.9	39.0
PWDEF	173.0	3.8	1.9	5.8

表 9-5　Virtex－Ⅱ对于领域的实现（布局后的）结果

电路名称	时钟（MHz）	领域				
		LUT	MULT	SRL16	FF	Slice
WDEF	25.6	286	8		56	161
PWDEF	173.0	430	8	144	464	448

9.5　RIS 硬件共享（调度算法）的实现

第 9.4 节提出了新 IRIS 工具的主要功能，它让设计者能够快速并且自动地从分层算法的 SFG 表达式中综合一个电路架构，并且评估综合架构上的时延和数据时序格式的影响。在有关架构的开发中，设计者也许想用硬件共享来研究解决之法。这需要复杂的控制，因此调度和产生电路的控制方式是必要的。本节阐述了 IRIS 工具中的调度算法和控制器，也就是分层调度（Yi 等 2002）的扩展 Minnesota 架构（Extended Minnesota Architecture Synthesis，EMARS）调度算法。这是对由开发了内部和彼此迭代优先级限制（Wang 和 Parhi 1994）提出的 MARS 调度算法的扩展。本节集中于 IRIS 的 EMARS 调度算法的采用，这是为了实现调度和硬件共享架构的转变。一个改进的折叠转换技术为硬件设计控制电路提供了一个系统化的技术，在这项技术中，几个算法操作都是时分复用到一个单个功能部件上的。

对 MARS 算法的主要改变是改变循环界限的计算来应用复杂组件和分层的电路。对最初的调度、解决冲突、调度和资源分配也需要相应的改变。折叠转换（Parhi 1999）为设计控制电路提供一个系统化的技术，其中几个算法操作都是时分复用到一个单个功能部件上的。折叠等式的推导是基于这个技术的。在分层的架构中，改变了折叠等式来适合分层的架构。EMRAS 算法的主要步骤如下所示。

步骤 1：递归节点的调度和资源分配算法

高级架构的综合目标是当使用现实技术的限制时，将一个算法的描述转换为一个有效率的架构设计。当满足速度和面积要求时，结果的架构一定保留了原始

的功能。正如之前所述，系统的递归部分决定了最大的采样速率。首先要考虑对递归节点的调度。

步骤 1.1：循环搜索和迭代界限

若指定了一个循环，则原始的 MARS 计算循环界限如下：

$$T_{\mathrm{LB}_j} = \frac{T_{\mathrm{L}_j}}{D_{\mathrm{L}_j}} \tag{9-8}$$

式中，T_{L_j} = 循环 j 的循环计算时间并且 D_{L_j} = 循环 j 中循环时延的数量。迭代界限能从循环界限值中通过确定最大的循环界限来计算。虽然原始的 MARS 仅仅研究所有输入都是同时输入，所有输出也都是同时输出的简单组件，但在这里研究了复杂和分层的架构模块的计算式。在 EMARS 中，关键路径的运算时间和循环界限的计算如下：

$$T_{\mathrm{LB}_j} = \frac{\sum_{e \in j} \mathrm{PWO}(e) - \sum_{e \in j} \mathrm{PWI}(e)}{\sum_{e \in j} \mathrm{SFGW}(e)} \tag{9-9}$$

$$T_{\mathrm{Crit}} = \sum_{e \in p} \mathrm{PWO}(e) - \sum_{e \in p} \mathrm{PWI}(e) \tag{9-10}$$

式中，PWO(e)、PWI(e) 和 SFGW(e) 的意义与式（9-6）给出的一样。j 和 p 的值分别指的是循环 j 和数据通道 p。

2 阶 IIR 滤波器被用于说明原始 MARS 调度算法的限制，并指出式（9-9）和式（9-10）对于 2 阶 IIR 滤波器是正确的。

看一下如图 9-13a 所示的包含 5 个 MAC 的 2 阶 IIR 滤波器的指标。据推测，每个乘法器和加法器拥有 1 个单元的运算时间，它与 FPGA 实现相当。MAC 的 MDP 模块如图 9-13b 所示，并且滤波器节点的关系图和 PGH 值如图 9-13c 所示。这个滤波器包含了一个拥有两个递归节点（MAC4 和 MAC5）的递归部分，以及一个拥有 3 个非递归节点的非递归部分（MAC1、MAC2 和 MAC3）。在这个滤波器中有两个循环：

循环 1：MAC5（P_{i}）→MAC5（S_{o}）→D_4

循环界限 $= \frac{2-0}{1} = 2$

循环 2：MAC4（P_{i}）→MAC（S_{o}）→MAC5（S_{i}）→MAC5（S_{o}）→D_4→D_3

循环界限 $= \frac{2-1+2-0}{1+1} = \frac{3}{2}$

关键路径：MAC1（S_{o}）→MAC2（S_{o}）→MAC3（S_{o}）→MAC4（S_{o}）→MAC5（S_{o}）

关键路径的运算时间 = 2 − 1 + 2 − 1 + 2 − 1 + 2 − 1 + 2 = 6

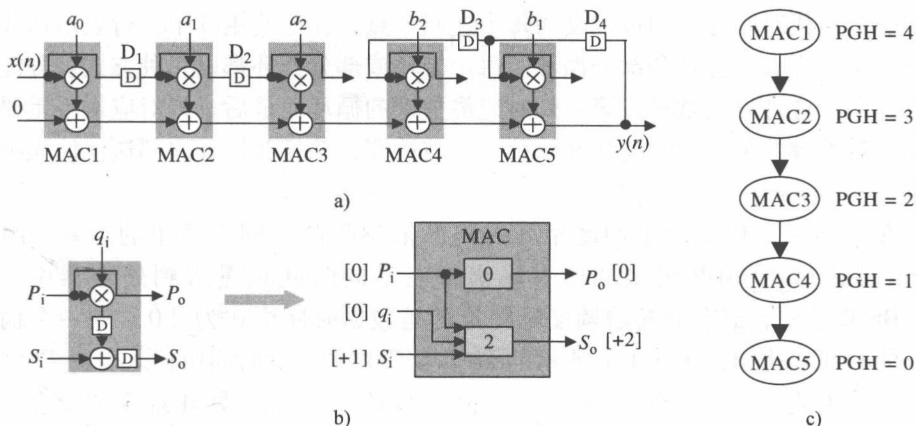


图9-13　a）使用MAC的2阶IIR滤波器的SFG；b）流水线MAC架构；c）依赖图和节点部分PGH值

摘自 *Hierarchical Synthesis of Complex DSP Functions Using IRIS* by Y. Yi & R. Woods，IEEE Trans on Computer Aided Design，Vol. 25，No. 5，© 2006 IEEE

因此需要用于调度递归节点的循环总数是2，即循环1和循环2。在滤波器中，迭代的界限是两个单元且关键路径的运算时间是6个单元。

步骤1.2　最初的调度

在步骤1.1中计算了迭代界限（或者流水线周期）。迭代时间分区被定义为时间步长，而执行任务的时间步长是模上迭代界限后的时间步长（Wang and Parhi 1994）。例如，2阶IIR滤波器的迭代界限是2，因此有两个迭代时间分区（0，1）。如果任务在时间步长为5上被调度，则这个任务被分配到5模2的时间分区1上。处理器在MARS中更低的界限给出如下：

$$(\text{Lowerbound})_u = \left[\frac{N_u \times T_u}{P_u \times T}\right] \tag{9-11}$$

式中，N_u = U型运算的数量，T_u = U型运算的运算时间，P_u = U型处理器的流水线重数且 T = 迭代周期。

与MARS调度算法中的单个运算相比，IRIS中分层架构中的不同输出会有不同的计算次数和流水线重数。在EMARS中，T_u 指的是一个U型运算的所有输出的最长计算时间，P_u 是一个U型运算的最高流水线级别。MAC的 P_u 和 T_u 与输出 S_o 相关。式（9-12）给出了在2阶IIR滤波器中MAC的更低界限，那意味着需要3个MAC处理器。

$$(\text{更低界限})_{MAC} = \left[\frac{5 \times 2}{2 \times 2}\right] = 3 \tag{9-12}$$

此时此刻，产生的调度矩阵对每种运算类型都有一个相应的矩阵。这些矩阵

表示一个迭代时间分区中被调度的每个递归节点，在这些矩阵中，行表示迭代时间分区，而列表示循环和部分循环。每个循环或部分循环都以一种方式被分配到一组列中，这种方式就是，第一列对应最关键的循环而最后一列对应最不重要的循环。最关键的循环从时间0开始第一个被调度，并且保持着内部迭代优先级的限制。

在IRIS中，用于产生调度矩阵的内部依赖限制不同于简单的运算，因为IRIS中改进后的MDP模型的所有输出都在不同时间点进入到处理器中。在EMARS算法中，节点的起始调度时间指的是数据时序格式为［0］的一个输入引脚的时间。比如，图9-13b所示的MAC处理器的调度时间由输入引脚P_i和Q_i决定。如果从处理器A到B有一个内部的依赖约束，那么处理器B的调度时间如下所示：

$$T_B = T_A + L_A - D_B \tag{9-13}$$

式中，T_A = 处理器A的调度时间，T_B = 处理器B的调度时间，L_A = 处理器A的输出引脚的时延，D_B = 处理器B的输入引脚的数据时序格式的最小值。

对于2阶IIR滤波器的例子，第一个调度关键的循环1，并且在时间0时调度节点MAC5。因为在循环2中MAC4和MAC5之间的内部依赖限制及MAC5的起始时间分区是0，通过使用式（9-13）将所有节点MAC4的调度时间计算为-1。MAC4是一个被包裹的节点（Wang和Parhi 1994），也就是一个在负的（也就是t）、大于（也就是$t<0$）或等于T（也就是$t>T$）的时间上被调度的节点。非包裹的节点是在时间步长等于时间分区（也就是$0 \leqslant t \leqslant T-1$）。包裹的节点在调度中在上方标记+1或-1。调度矩阵和最开始的调度见表9-6和表9-7，在里面L1和L2指的是循环且M1，M2和M3是物理的MAC处理器。

表9-6　对2阶IIR滤波器的调度矩阵

时间	L1	L2
0	MAC5	
1		$MAC4^{+1}$

表9-7　对2阶IIR滤波器的最初调度

时间	M1	M2	M3
0	MAC5		
1	$MAC4^{+1}$		

整个循环的节点可能会在违反彼此迭代优先级限制之前被转移，而这个转移的次数则由集合L的每个成员的循环灵活性定义。MARS通过下面的等式计算每个循环L的灵活性F，其中，T是迭代周期，D_L是循环时延的数量，T_L是循环运行时间。

$$F = T \times D_{\mathrm{L}} - T_{\mathrm{L}} \tag{9-14}$$

对于 EMARS，$T_{\mathrm{L}} = \sum_{e \in L}\mathrm{PWO}(e) - \sum_{e \in L}\mathrm{PWI}(e)$ 并且 $D_{\mathrm{L}} = \sum_{e \in L}\mathrm{SFGW}(e)$。

循环 1 和循环 2 的灵活性能被定为 $F_1 = 2 \times 1 - 2 = 0$ 且 $F_2 = 2 \times 2 - (2 + 2 - 1 - 0) = 1$。

步骤 1.3：解决冲突

为了解决冲突，EMARS 的算法采用了与 MARS 调度同样的原则（Wang and Parhi 1994）。在 2 阶滤波器的例子中，对递归节点一个有效的无冲突的调度与表 9-7 中所示的最初调度一样。

步骤 2：无递归节点的调度和资源分配

在这个子节中，无递归节点的调度和资源分配的主要 EMRAS 步骤都有讲解。

步骤 2.1：计算处理器的最小数量

通过使用式（9-15）能计算出在 MARS 中添加的处理器的确切数量。

$$(\text{处理器}) = \left\lceil \frac{(N_{\mathrm{u}} - TS_{\mathrm{u}}) \times T_{\mathrm{u}}}{P_{\mathrm{u}} \times T} \right\rceil \tag{9-15}$$

式中，N_{u} 是无递归部分的 U 型运算的数量，TS_{u} 是 U 型处理器中可利用的时间分区的数量，T_{u} 是 U 型运算的运算时间，P_{u} 是 U 型处理器的流水线重数并且 T 是迭代周期。

在 EMRAS 中，T_{u} 指的是一个 U 型运算所有输出的最大运算时间并且 P_{u} 是一个 U 型运算的流水线的最高重数。对于 2 阶 IIR 滤波器，添加的 MAC 处理器的确切数量给出如下：

$$(\text{处理器})_{\mathrm{MAC}} = \left\lceil \frac{(3 - 4) \times 2}{2 \times 2} \right\rceil = 0 \tag{9-16}$$

对于 MAC 处理器有 4 个时间分区，并且对于无递归 MAC 节点需要 3 个时间分区。MARS 决定了这个滤波器不需要任何新的处理器，因此需要的 MAC 处理器的数量是 3 个，这在式（9-12）中给出了。

步骤 2.2：定位前馈路径

图 9-13 所示的 2 阶 IIR 滤波器仅仅包含如图 9-14 所示的一个前馈电路，它的一端连接到一个递归节点。递归节点 MAC4 不包括在这个前馈路径中。

步骤 2.3：创建一个最初的计划表

继续研究图 9-13 所示的 2 阶 IIR 滤波器，来自前馈路径最初的计划表见表 9-8，在这个表格中，M1 ~ M3 指的是 3 个 MAC 处理器。注意到调度矩阵中没有冲突，并且它是无递归节点最后的无冲突调度。

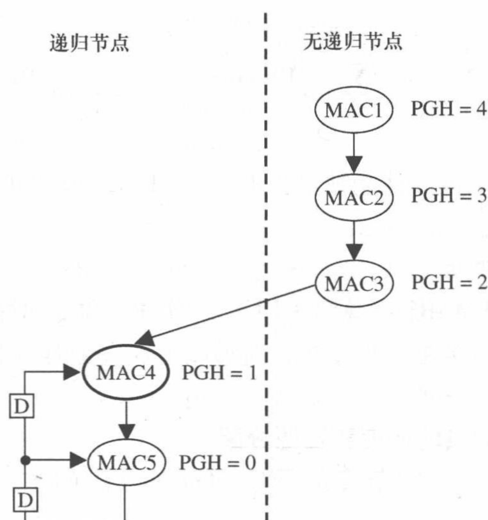


图 9-14　一个 2 阶 IIR 滤波器的计划表

步骤 3：使用改进的可折叠技术控制电路的实现

在给出最后的调度和处理器数量后，IRIS 能构建连接处理器的数据通道，并且使用一个改进的折叠转换技术产生控制电路。形成 IRIS 控制器基础的改进折叠等式的根源会在本节中得到介绍。

表 9-8　2 阶 IIR 滤波器的最后无冲突调度

时间	MAC 处理器		
	M1	M2	M3
0	MAC5	MAC3 $^{+1}$	MAC1 $^{+2}$
1	MAC4 $^{+1}$	MAC2 $^{+1}$	

当折叠转化减小架构中功能部件的数量时，也许会产生一个使用大量寄存器的架构。一些技术能被用于最小化寄存器数量（Parhi 1994），但是它们提高了分布电路的复杂度。在本节中，硬件共享电路是针对 Xilinx FPGA 的。虽然FPGA 是一个“寄存器富有”架构并且路由电路的互连时延能上升到与 FPGA 中总的关键路径的60%一样多，但也并没有故意减少寄存器数量，因为通过重定时程序可以使用这些时延来提高吞吐量。这些议题在下一节时延的 LMS 滤波器的实现中有讲述（Yi 和 Woods　2006，Yi 等　2005）。

步骤 3.1：改进的折叠转换

如第 8.6.2 节所述，一个折叠集合是由相同的功能部件实行运算的一个有序集合（Parhi 1999）。每个折叠集合包含 N 个元素，它们其中一些也许是无效的操作。在折叠集合中（在里面 j 从 0 到 $N-1$），第 j 个位置的操作通过在时间分区 j 期间的功能部件来执行。系统的折叠技术的使用是通过使用 2 阶 IIR 滤波器

的例子来演示的。以折叠因数 $N=2$ 来折叠这个滤波器，它使用的折叠集合为 S1 = MAC5，MAC4，S2 = MAC3，MAC2 以及 S3 = MAC1。折叠集合 S1，S2 及 S3 仅仅包含了 MAC 运算，并且在相同折叠集合中的节点通过同样的 MAC 硬件被执行。折叠因数 $N=2$ 意味着折叠硬件的迭代周期是 2。

如果能实现一个折叠的系统，那么 $D'_F(U \xrightarrow{e} V) \geqslant 0$ 必然适合所有 SFG 中的边缘。一旦有效的折叠集合被分配，那重定时要么能被用于满足这个性能，要么用于证明折叠集合不可行。通过使用式（9-17），找到了 SFG 每个箭头的一组约束条件，并且为了解决不等式的系统，使用这项技术来决定是否存在一个解决之法并找到一个解决之法，前提是解决之法存在。不等式限制给出如下：

$$r(U) - rV \leqslant \left[\frac{D'_F(U \xrightarrow{e} V)}{N}\right] \tag{9-17}$$

式中，$[x]$ 是小于等于 x 的最大整数。

下面将讲述一种方法，通过这种方法，这些技术能被用于设计折叠架构并产生折叠的 2 阶 IIR 滤波器。基本程序如下：

1）执行折叠的重定时；

2）写出折叠等式；

3）执行寄存器分配；

4）画出折叠的架构。

折叠等式（见式（8-10））和在图 9-13a 中的 SFG 的折叠限制（见式（9-17））的重定时见表 9-9。根据求解不等式系统的技术，第一步是画出约束图。考虑到有 N 个变量 M 组不等式，其中每个不等式都有 $r_i - r_j \leqslant k$ 的形式且 k 为整数，约束图使用以下程序画出（Parhi 1999）：

1）为 N 个变量画出节点 i；

2）为每个不等式 $r_i - r_j \leqslant k$，画出边缘 $j \to i$，以长度为 k，从节点 j 到节点 i；

3）为每个节点 i，$i=1, 2, \cdots, N$，画出节点 $N+1$，画出 $N+1 \to i$，以长度为 0 从 $N+1$ 到节点 i 的箭头。

表 9-9 为合拢约束的合拢等式及重定时

边缘	合拢等式	合拢等式的重定时
$1 \to 2(P)$	$D'_F(U \xrightarrow{e} V) = 2(1) - 0 + 0 + 1 - 0 = 3$	$r(1) - r(2) \leqslant 1$
$1 \to 2(S)$	$D'_F(U \xrightarrow{e} V) = 2(0) - 2 + 1 + 1 - 0 = 0$	$r(1) - r(2) \leqslant 0$
$2 \to 3(P)$	$D'_F(U \xrightarrow{e} V) = 2(1) - 0 + 0 + 0 - 1 = 1$	$r(2) - r(3) \leqslant 0$
$2 \to 3(S)$	$D'_F(U \xrightarrow{e} V) = 2(0) - 2 + 1 + 0 - 0 = -2$	$r(2) - r(3) \leqslant -1$
$3 \to 4(S)$	$D'_F(U \xrightarrow{e} V) = 2(0) - 2 + 1 + 1 - 0 = 0$	$r(3) - r(4) \leqslant 0$
$4 \to 5(S)$	$D'_F(U \xrightarrow{e} V) = 2(0) - 2 + 1 + 0 - 1 = -2$	$r(4) - r(5) \leqslant -1$
$5 \to 5(S)$	$D'_F(U \xrightarrow{e} V) = 2(1) - 2 + 0 + 0 - 0 = 0$	$r(5) - r(5) \leqslant 0$
$5 \to 4(P)$	$D'_F(U \xrightarrow{e} V) = 2(2) - 2 + 0 + 1 - 0 = 3$	$r(5) - r(4) \leqslant 1$

使用最短路径算法（Parhi 1999）之一能确定是否存在一个解决之法并且找到它。如果或者仅当约束图没有负循环时，不等式的系统才有一个解决之法。如果解决之法存在，那么它是使得从 $N+1$ 到节点 i 的最小距离 r_i 的方法。在表9-9的右列中，不等式的约束图如图9-15所示。即便有这一系列的约束，但也是能够求解的，因为如果没有负循环存在，则这个解是 $r(1)=-2$，$r(2)=-2$，$r(3)=-1$，$r(4)=-1$，$r(5)=0$。

重定时的 SFG 如图9-16所示。重定时的 SFG 的折叠等式见表9-10，折叠的 SFG 如图9-17所示。

由于将 DSP 算法映射到时分复用架构，因此在本节介绍了改进的折叠转换。新的折叠等式对分层的和平坦的电路都合适。对平坦电路而言，A_v将等于0，而折叠的等式则是最原始的那个等式。

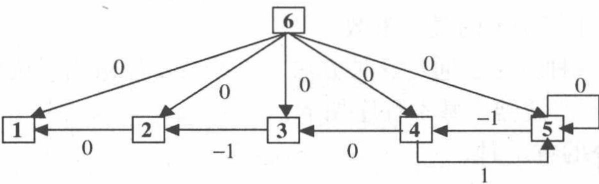


图9-15　2阶 IIR 滤波器的约束图形

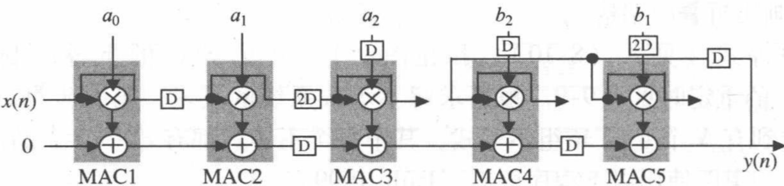


图9-16　重定时的2阶 IIR 滤波器。摘自 *Hierarchical Synthesis of Complex DSP Functions Using IRIS* by Y. Yi & R. Woods，IEEE Trans on Computer Aided Design，Vol. 25，No. 5，© 2006 IEEE

表9-10　图9-16中重定时 SFG 的合拢等式

边缘	合拢等式
1→2(P)	$D'_F(U \xrightarrow{e} V)=2(1)-0+0+1-0=3$
1→2(S)	$D'_F(U \xrightarrow{e} V)=2(0)-2+1+1-0=0$
2→3(P)	$D'_F(U \xrightarrow{e} V)=2(2)-0+0+0-1=3$
2→3(S)	$D'_F(U \xrightarrow{e} V)=2(1)-2+2+0-1=0$
3→4(S)	$D'_F(U \xrightarrow{e} V)=2(0)-2+1+1-0=0$
4→5(S)	$D'_F(U \xrightarrow{e} V)=2(1)-2+1+0-1=0$
5→5(S)	$D'_F(U \xrightarrow{e} V)=2(1)-2+0+0-0=0$
5→4(P)	$D'_F(U \xrightarrow{e} V)=2(1)-2+0+1-0=1$

（续）

边缘	合拢等式
$a_0 \to 1(Q)$	$D'_F(U \xrightarrow{e} V) = 2(0) - 0 + 0 + 0 - 0 = 0$
$a_1 \to 2(Q)$	$D'_F(U \xrightarrow{e} V) = 2(0) - 0 + 0 + 1 - 0 = 1$
$a_2 \to 3(Q)$	$D'_F(U \xrightarrow{e} V) = 2(1) - 0 + 0 + 0 - 0 = 2$
$b_2 \to 4(Q)$	$D'_F(U \xrightarrow{e} V) = 2(1) - 0 + 0 + 1 - 0 = 3$
$b_1 \to 5(Q)$	$D'_F(U \xrightarrow{e} V) = 2(2) - 0 + 0 + 0 - 0 = 4$

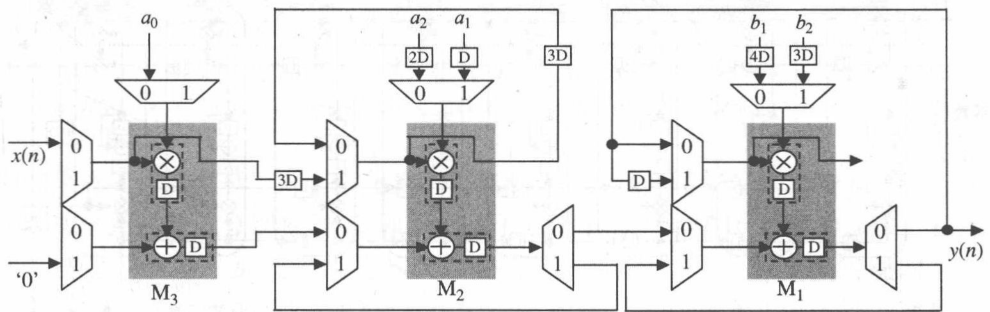


图 9-17 合拢的 2 阶 IIR 滤波器。摘自 *Hierarchical Synthesis of Complex DSP Functions Using IRIS* by Y. Yi & R. Woods，IEEE Trans on Computer Aided Design，Vol. 25，No. 5，© 2006 IEEE

9.6 实例研究：自适应时延最小均方的实现

自适应滤波器已经有许多应用，不过这些应用都需要不同的滤波器特性来响应变化的信号状态，比如在噪声消除、线性预测、自适应信号加强和自适应控制的应用中都有使用。在实践中，LMS 算法在实践中是使用最广泛的自适应滤波算法（Farhang-Boroujeny 1998）。LMS 算法的宽频应用是对于信号统计来说，它比较简单而且稳健。TF-DLMS 算法由以下等式描述。

滤波器输出：

$$y_n = \sum_{i=0}^{N-1} \omega_{n-i}, x_{n-i} = \omega_{n,0} x_n + \omega_{n-1,1} x_{n-1} + \cdots + \omega_{n-(N-1),N-1} x_{n-(N-1)} \tag{9-18}$$

误差：

$$e(n-m) = d(n-m) - y(n-m) \tag{9-19}$$

时延的系数更新：

$$\omega_{n,i} = \omega_{n-1,i} + 2\mu e_{n-m} x_{n-m-i} \tag{9-20}$$

一个 8 抽头的 TF－DLMS 架构如图 9-18 所示。使用 DLMS 算法，能在适应环路之前将寄存器嵌入到失真反馈路径中。现在主要的困难是将这些时延用作将 LMS 滤波器流水线化的一种方法。这个过程首先涉及时延数量的确定，这些时延是对实现一个彻底的流水线类型电路很必要的。这个数量值很重要，太小的值会产生一个低速电路，同时太大的值会产生一个更低的收敛速率并导致跟踪能力较差。IRIS 根据电路性能要求（速度和面积）确定了未知的时延数量（比如 mD）。一旦确定，就能开发一个彻底的流水线实现。

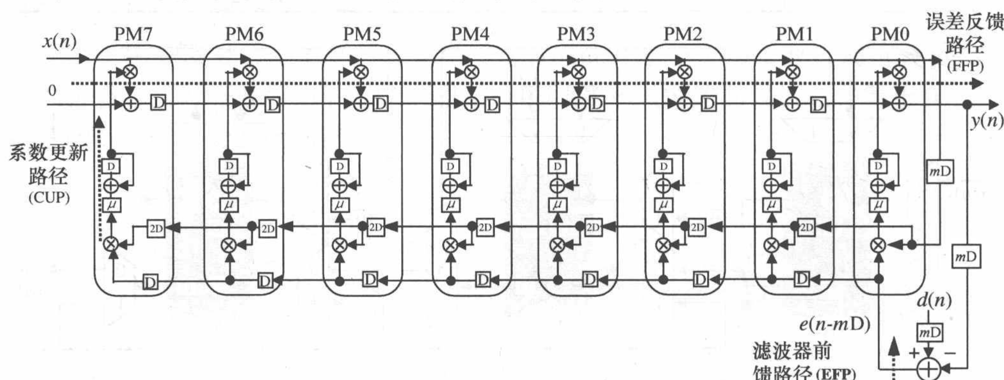


图 9-18 一个 8 抽头 TF－DLMS 算法架构

9.6.1 高速实现

TF－LMS 自适应滤波器（Jones 1992）有着与 DLMS 滤波器相似的收敛行为。一个有着 mD 时延的 8 抽头预测器系统如图 9-18 所示。时延的数量已经定义为 mD，因为这个值会通过还没有被确定的流水线站（mD）确定下来。这强调了在电路架构层的选择是怎样影响算法设计的。

在 EEP 数据通道中 m 时延的 8 抽头 TF－DLMS 的分层 SFG 如图 9-19 所示。它显示了 DLMS 滤波器是根据一个加法器和 8 个抽头单元来构建的。在这些抽头单元中，一个 MAC 和几个 MACμ（一个 MAC 和一个放缩电路 μ 一起）运算合并起来实现乘法器和累加功能。

一个电路算法的表达式是一个三层的分层电路。被选作这个设计例子的处理器类型包括了流水线式的专用乘法和加法处理器。通过使用定点运算能实现 DLMS 滤波器，并且考虑了所有的定点实现问题（Ting 等 2000）。这个分析包括为了根据自适应滤波器性能及实现的成本来实现最佳的字长的研究，截断/凑整电路的应用，饱和度的使用和简单乘法移位运算的开发（Ting 等 2000）。

当针对的是一个 FPGA 的实现时，我们假定给出的规范需要一个高速的电路，那么乘法器和加法器会被流水线化。在这个设计中，MAC 和 MACμ 子系统

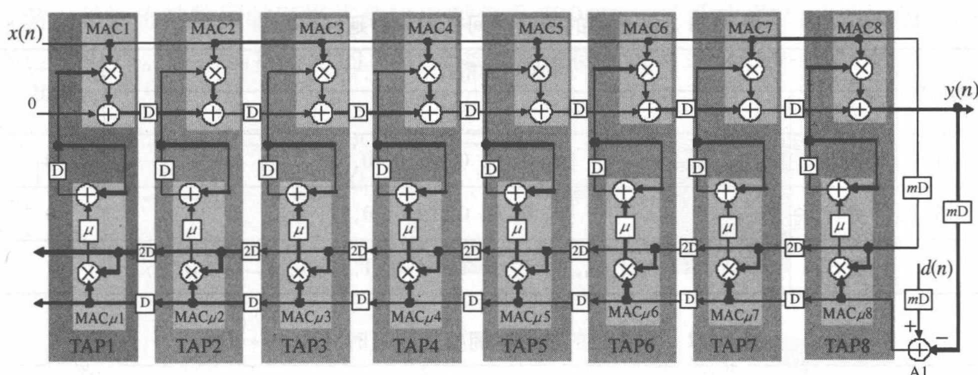


图 9-19　折叠的 2 阶 IIR 滤波器。摘自 *Hierarchical Synthesis of Complex DSP Functions Using IRIS* by Y. Yi & R. Woods，IEEE Trans on Computer Aided Design，Vol. 25，No. 5，© 2006 IEEE

中的乘法器运算是基于一个被化为一重流水线的专用 8 位符号乘法器。对加法处理器的电路是基于分别在 MAC 和 MACμ 子系统中的 8 位和 16 位字的加法，并且通过使用一个被一重流水线化的 5 位快速进位加法器，这是能够实现的。在 RDLMS 设计中使用的步长是 2^{-5}（0.03125），通过使用 Matlab™ 仿真，这是最接近最佳步长的 2 次幂数。

将乘法器和加法器的时延改变为一个时延会产生如图 9-20 的电路架构。在 IRIS SignalCompiler 读取了 8 抽头的 TF－DLMS 自适应滤波器的模型文件后，产生了基本参数化的处理器模型的图解表达式，这个表达式显示出了包含在 IRIS 处理器库中的信息，如图 9-20 所示。所有输入和输出数据时序格式及用于这个设计中的基本 Xilinx 处理器模块的输入时延见表 9-11 和表 9-12。需要提供图 9-21中节点的参数值来计算滤波器设计的具体时延值和数据时序模型。在这个例子中，通过使用 9.4 节中描述的 IRIS 工具能够实现分层设计的自动综合过程。

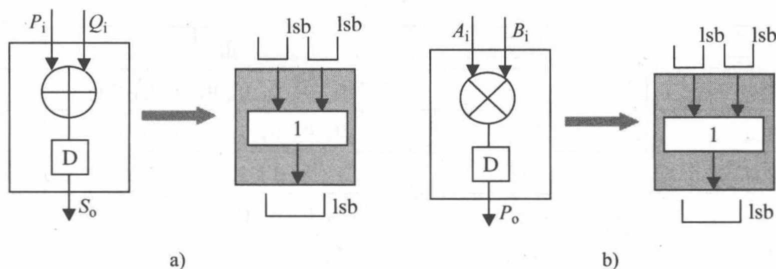


图 9-20　乘法器和加法器模型。摘自 *Hierarchical Synthesis of Complex DSP Functions Using IRIS* by Y. Yi & R. Woods，IEEE Trans on Computer Aided Design，Vol. 25，No. 5，© 2006 IEEE

a）乘法器　b）加法器

表 9-11 乘法器的数据时间格式和时延（$S = A * B$）

端口名	数据时间格式		时延
	类型	值	
A	并行	[0, 0, 0, 0, 0, 0, 0, 0]	0
B	并行	[0, 0, 0, 0, 0, 0, 0, 0]	0
S	并行	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]	1

表 9-12 加法器的数据时间格式和延时（$S = A + B$）

端口名	数据时间格式		时延
	类型	值	
A	并行	[0, 0, 0, 0, 0, 0, 0, 0]（8 位） [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]（16 位）	0
B	并行	[0, 0, 0, 0, 0, 0, 0, 0]（8 位） [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]（16 位）	0
S	并行	[1, 1, 1, 1, 1, 1, 1, 1]（8 位） [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]（16 位）	1

由于在 MAC 和 MACμ 的子系统中没有了反馈环，因此流水线周期就变成了1。通过使用第9.4 节中描述的重定时技术，IRIS 能自动综合 MAC 和 MACμ 子系统，并且通过使用这种方法，将截断问题也纳入了考虑。MAC 和 MACμ 子系统的原始电路、综合电路和 MDP 模型分别如图 9-21a 和图 9-21b 所示。所有的输入和输出数据时序格式和输出时延见表 9-13 和表 9-14。相关的输入向量见表9-15。

表 9-13 MAC 子系统的数据时间格式和时延

端口号	数据时间格式		时延
	类型	值	
P_i	并行	[0, 0, 0, 0, 0, 0, 0, 0]	
Q_i	并行	[0, 0, 0, 0, 0, 0, 0, 0]	
S_i	并行	[1, 1, 1, 1, 1, 1, 1, 1]	
P_o	并行	[0, 0, 0, 0, 0, 0, 0, 0]	0
S_o	并行	[2, 2, 2, 2, 2, 2, 2, 2]	2

子系统 MAC 和 MACμ 产生的 MDP 模型用在了高级子系统的综合中。高级子系统是一个 TAP 子系统且它的流水线周期为 1。

TAP 子系统的原始电路、综合电路和 MDP 模型如图 9-21c 所示。TAP 的所

有输入和输出的数据时序格式和输出时延见表 9-16。子系统 TAP 的相关输入向量见表 9-15。包括函数时延的 TAP 子系统的数据通道时延见表 9-17。

图 9-21　TF－DLMS 滤波器中的子系统模型。摘自 *Hierarchical Synthesis of Complex DSP Functions Using IRIS* by Y. Yi & R. Woods, IEEE Trans on Computer Aided Design, Vol. 25, No. 5, © 2006 IEEE

a）MAC 子系统的 MDP 模型的创建　b）MAC μ 子系统的 MDP 模型的创建

c）TAP 子系统的 MDP 模型的创建

表 9-14　MAC 的数据时间格式和延时

端口号	数据时间格式		时延
	类型	值	
P_i	并行	[0, 0, 0, 0, 0, 0, 0, 0]	
Q_i	并行	[0, 0, 0, 0, 0, 0, 0, 0]	
S_i	并行	[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]	
P_o	并行	[0, 0, 0, 0, 0, 0, 0, 0]	0
Q_o	并行	[0, 0, 0, 0, 0, 0, 0, 0]	0
S_o	并行	[2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2]	2

表 9-15 MAC、MAC 和 TAP 的相关输入向量

模块输出	MAC	MAC	模块输出	TAP
P_o	$[P_i]$	$[P_i]$	x_o	$[x_i]$
Q_o		$[Q_i]$	x_{ol}	$[x_{il}]$
S_o	$[P_i \quad Q_i \quad S_i]$	$[P_i \quad Q_i \quad S_i]$	e_o	$[e_i]$
			y_o	$[x_i, x_{il}, e_i, y_i]$

表 9-16 TAP 的数据时间格式和延时

端口号	数据时间格式		时延
	类型	值	
x_i	并行	[1, 1, 1, 1, 1, 1, 1, 1]	
x_{il}	并行	[0, 0, 0, 0, 0, 0, 0, 0]	
e_i	并行	[0, 0, 0, 0, 0, 0, 0, 0]	
y_i	并行	[2, 2, 2, 2, 2, 2, 2, 2]	
x_o	并行	[1, 1, 1, 1, 1, 1, 1, 1]	1
x_{ol}	并行	[0, 0, 0, 0, 0, 0, 0, 0]	0
e_o	并行	[0, 0, 0, 0, 0, 0, 0, 0]	0
y_o	并行	[3, 3, 3, 3, 3, 3, 3, 3]	3

表 9-17 TAP 子系统的数据通道延时对

	TAP			
	x_i	x_{il}	e_i	y_i
x_o	(0, 0)			
x_{ol}		(0, 0)		
e_o			(0, 0)	
y_o	(2, 0)	(4, 1)	(4, 1)	(0, 0)

通过使用 TAP 子系统和加法器处理器的 MDP 模型，综合了高级的 8 抽头 TF－DLMS 电路。插入到 TF－DLMS 电路的失真反馈（mD）中的流水线数量需要确定下来。如下所示的高级 8 抽头 TF－DLMS 架构中识别出了 8 个环路。通过使用式（9-16），能够计算出相应的流水线周期。确定了 1～8 的流水线周期并且使之等于 1，都是为了产生一个高速架构。这些环的最佳值为：$m_8=-3$D，$m_7=-2$D，$m_6=-1$D，$m_5=0$D，$m_4=1$D，$m_3=2$D，$m_2=3$D，$m_1=4$D。从这些等式中，能够看到通过最慢的循环确定出整个架构的时延数量的最佳值 m 等于 4D。通过使用 IRIS，8 抽头预测滤波器的综合电路如图 9-22 所示，其中综合的抽头架构如图 9-21c 所示。

循环 1：TAP8（y_o）→mD→A1→TAP8（e_i）

$$\alpha_1 = \left\lceil \frac{4+1-0-0}{m+1} \right\rceil = 1 \Rightarrow m_1 = 4\text{D}$$

循环2：TAP7（y_o）→TAP8（y_o）→mD→A1→TAP8（e_o）→TAP7（e_i）

$$\alpha_2 = \left\lceil \frac{4+4+1+0-2-0-0-0}{1+1+m+2} \right\rceil = 1 \Rightarrow m_2 = 3\text{D}$$

循环3：TAP6（y_o）→TAP7（y_o）→TAP8（y_o）→mD→A1→TAP8（e_o）→TAP7（$\bar{e}_o$）T→AP6（e_i）

$$\alpha_3 = \left\lceil \frac{4+4+1+0+0-2-2-0-0-0-0}{1+1+1+m+4} \right\rceil = 1 \Rightarrow m_3 = 2\text{D}$$

⋮

循环8：TAP1（y_o）→TAP2（y_o）→TAP3（y_o）→AP4（y_o）→TAP5（y_o）→TAP6（y_o）→TAP7（y_o）→TAP8（y_o）→mD→A1→TAP8（e_o）→TAP7（e_o）T→TAP6（e_o）→TAP5（e_o）→TAP4（e_o）T→TAP3（e_o）→TAP2（e_o）→TAP1（e_i）

$$\alpha_8 = \left\lceil \frac{4\times8+1+0\times7-0-0\times6}{1\times8+m+14} \right\rceil = 1 \Rightarrow m_8 = -3\text{D}$$

$m = \max$（allm_c） $=4$D

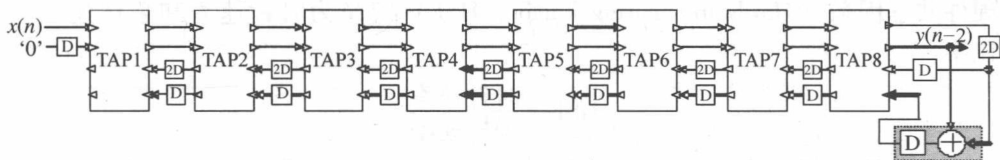


图9-22　综合的 TF－RDLMS 滤波器。摘自 *Hierarchical Synthesis of Complex DSP Functions Using IRIS* by Y. Yi & R. Woods，IEEE Trans on Computer Aided Design，Vol. 25，No. 5，© 2006 IEEE

通过使用如图9-23所示的 TAP 处理器模块结构，能构建出 TF－RDLMS N 抽头滤波器。在这个 TAP 结构中，会局部更新滤波器权重，因此如果需要，则可以通过添加更多的 TAP 元件，使序数随之增加。滤波器也有着优势，那就是增加滤波器的序数不会增加关键路径。被用于更新滤波器系数并影响滤波器性能的误差信号 $e(n)$ 的时延被增加了 $4N$ 个而不是如提供性能优势的 TF－FPDLMS 结构中的 $3N$ 个。

9.6.2　按具体性能要求的硬件共享设计

由于 Xilinx FPGA Virtex－Ⅱ技术的优势，专用加法器和乘法器的速度已经提升到了180MSPS。在大量应用中，采样速率能远超预期，因此硬件共享是值得开发的。为了实现硬件共享，可以使用 EMARS 调度算法和改进的折叠技术。在这里，通过使用 IRIS 调度功能，能产生8抽头 TF－DLMS 滤波器的硬件共享电路。

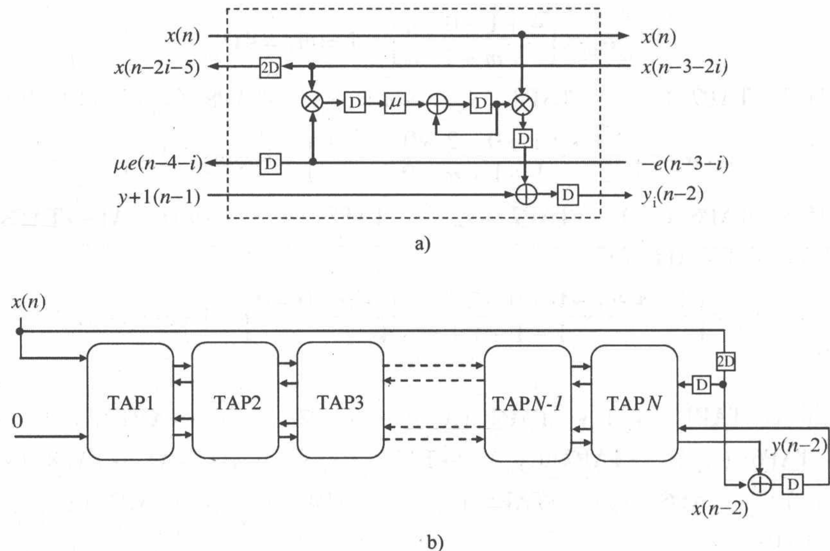


图 9-23　被调换的细粒重定时 DLMS 滤波器
a）TAP 模型　b）N 阶 TF－RDLMS 滤波器

硬件共享因数（Hardware Sharing Factor，HSF）定义为时钟速率和采样速率的比率：

$$\mathrm{HSF}=\frac{\text{时钟速率}}{\text{采样速率}}$$

它也等于流水线周期。假定电路采样速率是 80MHz，那么 HSF 就是 2，因为 ⌈180/80⌉ =2。

因为 MAC 和 MAC 的子系统不包括反馈环和时延，因此流水线周期为 2 的综合电路和 MDP 模型与图 9-21a 和 9-21b 所示的流水线周期为 1 的电路一样。TAP 子系统的综合电路及相应的流水线周期为 2 的 IRISMDP 模型如图 9-24 所示。所有的输入和输出数据时序格式及流水线周期为 2 的 TAP 输出时延见表 9-18。TAP 子系统的相关输入向量与表 9-15 中的 TF－DLMS 滤波器相同。拥有共享电路的 TAP 子系统的数据通道时延见表 9-19。

通过使用 TAP 子系统 MDP 模型和加法处理器，能够综合高级 8 抽头的 TF－DLMS 电路。在误差反馈（mD）中需要确定插入的流水线数量。图 9-19 的TF－DLMS 电路使用了同样的方法来确定最优化值 m。在这种情况下，最优化值 m 是 2，因为流水线周期是 2。对于 TF－DLMS，最后的无冲突调度矩阵见表 9-20。注意到迭代周期在这个矩阵中是等于流水线周期的。但是，因为严格的优先级约束，所以分配了一个新的处理器。在这个例子中，最后的硬件表示符的数量确切地等于 4 个 TAP 表示符（T1～T4）和一个加法表示符（A1），见表 9-20。

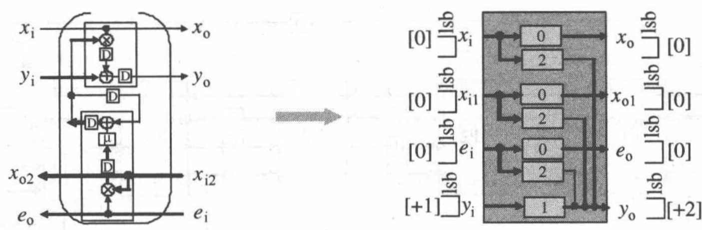


图 9-24 综合的抽头子系统和 MDP 模型（$\alpha=2$）

表 9-18 TAP 的数据时间格式和时延

端口名	数据时间格式		时延
	类型	值	
x_i	并行	[0, 0, 0, 0, 0, 0, 0, 0]	
x_{il}	并行	[0, 0, 0, 0, 0, 0, 0, 0]	
e_i	并行	[0, 0, 0, 0, 0, 0, 0, 0]	
y_i	并行	[1, 1, 1, 1, 1, 1, 1, 1]	
x_o	并行	[0, 0, 0, 0, 0, 0, 0, 0]	0
x_{ol}	并行	[0, 0, 0, 0, 0, 0, 0, 0]	0
e_o	并行	[0, 0, 0, 0, 0, 0, 0, 0]	0
y_o	并行	[2, 2, 2, 2, 2, 2, 2, 2]	2

表 9-19 TAP 子系统的数据通道时延对

	TAP			
	x_i	x_{il}	e_i	y_i
x_o	(0, 0)			
x_{ol}		(0, 0)		
e_o			(0, 0)	
y_o	(2, 0)	(4, 2)	(4, 12)	(1, 0)

表 9-20 一个硬件共享的 TF - DLMS 滤波器的调度矩阵

时间步长	TAP				加法器
	T1	T2	T3	T4	A1
0	TAP1	TAP2	TAP3	TAP4	加法器
1	TAP5	TAP6	TAP7	TAP8	

通过使用 IRIS 调度功能，流水线周期为 2 的 TF - DLMS 的硬件共享电路如图 9-25 所示。

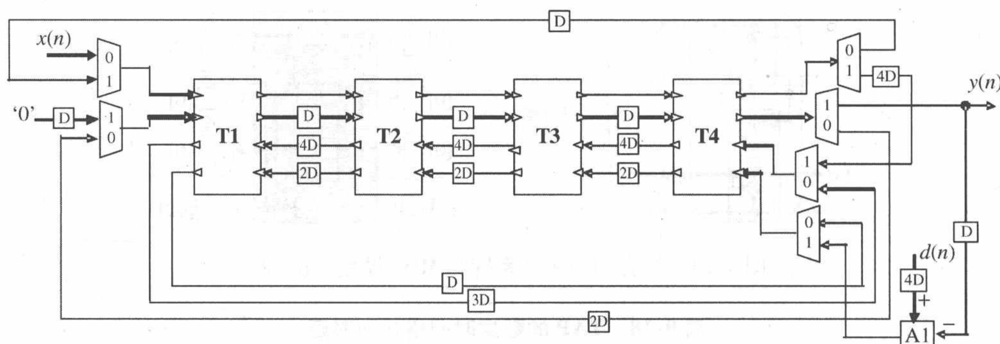


图 9-25 如同由流水线周期为 2 的 IRIS 生成图 9-5 的 TF - DLMS 滤波器的最后硬件架构

9.7 总结

本章描述了产生一个基于 SFG 的综合工具 IRIS 的工作，许多技术自动化的 IRIS 已经在第 8 章中提到。此外，本章说明了设计的分层性在提到过的技术上因何存在相当大的设计限制。在本章中，有两个例子，也就是 2 阶 IIR 滤波器和 5 阶 WDE 滤波器已经用于克服这个困难。这使得我们对这个最初进行 FPGA 实现时便会遇到的困难有了清晰的认识。第 12 章将详细讲解一些额外的方面，以有用的 IP 核方式实现这个功能便需要这些方面。

参考文献

AccelFPGA (2002) Accelfpga: High-level synthesis for dsp design. Web publication downloadable from http://www.accelchip.com/.

Bellows P and Hutchings B (1998) Jhdl – an hdl for reconfigurable systems. *IEEE Symposium on FPGA's for Custom Computing Machines*, pp. 175–184, Napa, USA.

Bringmann O and Rosenstiel W (1997) Resource sharing in hierarchical synthesis. *Int. Conf. on Computer Aided Design*, pp. 318–325.

C. E. Leiserson FMR and Saxe JB (1983) Optimizing synchronous circuitry by retiming. *Proc. 3rd Caltech Conference on VLSI*, pp. 87–116.

Christofides N (1975) *Graph Theory – An Algorithmic Approach*, Academic Press, London.

Davidson S, Landskov D, Shriver B and Mallett PW (1981) Some experiments in local microcode compaction for horizontal machines. *IEEE Trans. Computers*, pp. 460–477.

Derrien S and Risset T (2000) Interfacing compiled fpga programs: the mmalpha approach. *Journal of Parallel and Distributed Processing Techniques and Applications*.

Dewilde P, Deprettere E and Nouta R (1985) *Parallel and Pipelined VLSI Implementation of Signal Processing Algorithms*, Prentice Hall, pp. 258–264.

Drost A 1999 Hierarchy management in synplify. Web publication downloadable from http://www.xilinx.com/.

Farhang-Boroujeny B (1998) *Adaptive Filters Theory and Applications*. John Wiley & Sons Inc.

Gazsi L (1985) Explicit formulas for lattice wave digital filters. *IEEE Trans. Circuits and Systems* CAS-32, 68–88.

Gnizio JP (1985) *Introduction to Linear Goal Programming*. Sage Publications, Beverly Hills.

Jones DL (1992) Learning characteristics of transpose-form lms adaptive filters. *IEEE Trans. Circuits and Systems-II: Analog and Digital Signal Proc.* 39, 745–749.

Keating M and Bricaud P (1998) *Reuse Methodology Manual for System-On-A-Chip Designs*. Kluwer Academic Publishers, Norwell, MA, USA.

Kung SY (1988) *VLSI Array Processors*. Prentice Hall, Englewood Cliffs, NJ.

Lawson S and Mirzai A (1990) *Wave Digital Filters*. Ellis Horwood, New York.

Lee J, Hsu Y and Lin Y (1989) A new integer linear programming formulation for the scheduling problem in data-path synthesis *Proc. Int. Conf. on Computer Aided Design*, pp. 20–23.

McCanny JV, Ridge D, Hu Y and Hunter J (1997) Hierarchical vhdl libraries for dsp asic design *Proc Int. Conf. on Acoustics, Speech and Signal Processing*, Munich, pp. 675–678.

McGovern B (1993) *The Systematic Design of VLSI Signal Processing Architectures*. PhD dissertation School of Electrical and Electronic Engineering, Queen's University of Belfast.

Parhi KK (1994) Calculation of minimum number of registers in arbitrary life time chart. *IEEE Trans. Circuits and Systems-II* 41(6), 434–436.

Parhi KK (1999) *VLSI digital signal processing systems : design and implementation*. John Wiley & Sons, Inc., New York.

Park IC and Kyung CM (1991) Fast and near optimal scheduling in automatic data path synthesis *Proc. 28th DAC*, pp. 680–685.

Paulin PG and Knight JP (1989) Force directed scheduling for the behavioural synthesis of asic's. *IEEE Trans. Computer Aided Design* 8, 661–679.

Roy J (1993) *Parallel Algorithms For High-Level Synthesis*. PhD dissertation the University of Cincinnati.

Synplify (2003) Synplify pro: the industry #1 fpga synthesis solution. Web publication downloadable from http://www.synplicity.com/.

Ting L, Woods R, Cowan C, Cork P and Sprigings C (2000) High-performance fine-grained pipelined lms algorithm in virtex fpga *Advanced Signal Processing Algorithms, Architectures, and Implementations X: SPIE San Diego*, pp. 288–299.

Trainor DW (1995) *An Architectural Synthesis Tool for VLSI Signal Processing Chips*. PhD dissertation School of Electrical and Electronic Engineering, Queen's University of Belfast.

Trainor DW, Woods RF and McCanny JV (1997) Architectural synthesis of digital signal processing algorithms using iris. *Journal of VLSI Signal Processing* 16(1), 41–56.

Vajda S (1981) *Linear Programming: Algorithms and Applications*. Chapman and Hall, London.

Vanhoof J, Rompaey KV, Bolsens I, Goossens G and De Man H (1993) *High-Level Synthesis for Real-Time Digital Signal Processing*. Kluwer Academic Publishers, Dordrecht/Boston/London.

Wang CY and Parhi KK (1994) The MARS High-Level DSP Synthesis System. *VLSI Design Methodologies for Digital Signal Processing Architectures*, Kluwer.

Xilinx Inc. (1999) Using xilinx and synplify for incremental designing (ceo). Web publication downloadable from http://www.xilinx.com/xapp/xapp164.pdf. Xilinx Application Notes.

Xilinx Inc. (2000) Xilinx system generator v2.1 for simulink reference guide. Web publication downloadable from http://www.xilinx.com.

Xilinx Inc. (2001) Xilinx software manual: Design manage / flow engineer guide. Web publication downloadable from http://toolbox.xilinx.com/docsan.

Yi Y and Woods R (2006) Hierarchical synthesis of complex dsp functions using iris. *IEEE Trans. Computer Aided Design*, 25(5), 806–820.

Yi Y, Woods R, Ting L and Cowan C (2002) Implementing high-speed delayed-lms filter using the virtex-ii fpg *Proc. IEE Non-Linear and Non-Gaussian Signal Processing - N2SP Workshop*, Peebles, London.

Yi Y, Woods R, Ting L and Cowan C 2005 High speed fpga-based implementations of delayed-lms filters. *J. VLSI Signal Processing* 39(1-2), 113–131.

第 10 章　FPGA 的复杂 DSP 核的设计

现阶段的硅芯片技术可以使数以百万计的逻辑门集成在 $1cm^2$ 的硅上，预计到 2015 年每 cm^2 的芯片将会容纳超过两百万的晶体管。现在可实现非常复杂的功能，将以前作为一个个体的芯片建成一个完整的 SoC。将一个应用的所有部分整合在同一块芯片上呈现出了更低功耗、更高可靠性和降低成本的优点。由此一来，与日俱增的压力迫使设计师面对不断缩短的，以月计算而不是以年计算的上市时间期限。用户市场的重点就是在一个短小的生命周期及更短的上市时间之内销售大量的廉价产品。尤其是当一个新的标准出现时，例如驱动了高清电视和移动视频发展的 H. 264 和 MPEG4。

不断增大的芯片容量使一些设备实现了 FPGA 功能的扩展，例如 Xilinx 的 Virtex V 和 Altera 公司的 Stratix III 就提供了全部 SoC 功能。这些大量可用的逻辑门带来了在这些设备上实现开发更加复杂系统的问题。随着集成电路的发展，以及设计复杂性的提升，在每个芯片上可用逻辑门数量的增长率及每个芯片上逻辑门数量的差异实际上在被人为地扩大。正如第 1 章中指出的，这通常被称为设计生产率差距（IRTS 1999），强调了在设计产业中不断地改进就不会被闭合的一个分歧。然而，完整转变设计与实施有数百万逻辑门芯片的方法论的需要，将使设计师集中在更高抽象层次的设计。

随着芯片容量的增大，设计复杂性增加的速度更快，因为现在完整的系统设计由更多方面组成，并且可以和一系列的技术学科结合起来。更多的工作在系统层面上，使设计师更加积极地参与集成关键组件而无暇深入钻研设计的功能性。现有的设计和验证方法进展不同步，导致了设计生产力和硅制造能力之间日益增大的差距。

在目前的电子设计中，测试和验证已成为备受关注的主要方面。随着芯片复杂性的难度成倍地增长，复杂系统的验证现已成为系统级设计的瓶颈。设计团队常常会花多达 90% 的开发工作量在难点或系统级验证上。很多研究中的对策是想开发系统来加速芯片的测试和验证，特别是解决在从第三方集成测试设计组件增加的困难。时间和金钱方面的顾虑则更加危如累卵。罗恩（2002）很好地总结了该行业的共识：

"分析师普遍认为在新产品开发项目过程中，更早、更快的硬件和软件的验证是降低风险的关键。"

本章将涉及可重复使用的设计过程的演变，重点是基于 FPGA 的 IP 核的生

成。同时针对如何重复使用 IP 核开发集成问题进行了讨论。首先，10.1 节列出可重用设计的动机并给一些重用目标；10.2 节将概述 IP 核的发展；10.3 节将强调如何从简单的算术库转变为复杂的系统组件；10.4 节将描述如何设定 IP 核参数，并使用 FIR 滤波器演示一些关键阶段；10.5 节是整合过程；10.6 节将描述 ADPCM IP 核的案例研究；10.7 节将简要说明第三方 FPGA IP 供应商；10.8 节是一些结论。

10.1 可重用设计的动机

提升设计和验证方法的迫切需要，将加快目前的设计过程，并使设计生产率的差距缩小。由于可用晶体管的数量在每一个技术周期都会翻倍，因此当前目标应该是以同样的速度提高设计效率。若达到这样的成就，则需要花费很多精力在研究设计、测试和经常被忽视的验证过程的机制。这预示着可重用设计是提高生产力，特别是帮助系统级设计的关键驱动因素之一。

除了成倍增加的晶体管数量，系统本身也变得越来越复杂，这是由于完整的系统在单一的设备上相结合，组件的异质性带来一系列的芯片设计问题，尤其是测试和验证问题。涉及全系统的设计意味着开发人员需要知道如何结合各种不同的组件建立一个完整的系统级设计。开发过程过于复杂影响到了设计效率，并造成了日益苛刻的上市时间期限。获得设计的平衡是一个多维难题，如性能、电源管理和可制造性，再加上上市时间和生产力，随着每项技术进步，其难度也在加大。

从最初的设计，到最终的功能测试和验证的整个项目开发过程中，采用可重用设计的策略能够提高设计生产率。通过提高抽象层次，设计团队可以用分层设计的方法专注于系统级设计的关键部件。这种方法需要全套的建模、仿真，以及组件和更高级模块的测试和验证。此验证需要一个能进行代码重用和功能测试覆盖的正式策略（Huang 等　2001，Moretti 2001，Varma 和 Bhatia 1998）。

为了提高整体生产力并与每一代技术保持同步，一个系统的重用设计的数量必须以同样的速度增加，抽象层次也必须提高。采用高级功能模块的重用策略的生产力收益预计将超过 200%（半导体行业协　2005）。可重用的组件需要预先有自己独立的、可以纳入到更高级别测试环境的测试工具。这可以从旧式设计或第三方供应商纳入 IP 核来实现。对这种核的需要推动了 IP 核市场的成长，更大比例的芯片元件来自于 IP 核。

在国际半导体技术发展路线图 2005 报告（半导体行业协会 2005）中，可重复使用的逻辑模块的比例目前是 36%（2008），这个数字到 2015 年预计将稳步增加到 48%，并在超越这个时限后以类似的方式继续增加。ITRS 曾发表一份更

新到 2005 年的路线图，给出了市场主导的驱动力（半导体行业协会 2006）。表 10-1列出了一些需要重用数据的摘要。

表 10-1　SOC 设计生产力趋势

	2005	2008	2010	2012	2014	2016	2018	2020
需要重用的设计（%）	30	42	50	58	66	74	82	90
趋势：SoC 总逻辑规模	1.0	2.2	3.4	5.5	8.5	13.8	20.6	34.2
新设计所需的生产力	1.0	2.0	3.0	4.6	6.7	10.2	14.3	22.1
重复使用所需的生产力设计	2.0	4.0	6.0	9.2	13.5	20.4	28.6	44.2

提高设计效率的另一个重要方面是构成与嵌入式软件原则相同的设计，从而可以集成到系统级设计的函数库。

10.2　IP 核

提高生产率最有利的解决方案之一，是使用预先设计好的被称为硅 IP 核的功能模块，它也常被称为虚拟电路（Virtual Circuit，VC）。IP 核的术语适用于各种应用，从专用电路布局设计到针对可编程 DSP 或 RISC 处理器的高效代码，或以 HDL 捕获的核心描述。在 ASIC 和 FPGA 应用的领域内，IP 内核常常被划分为三类，即硬 IP；固 IP 和软 IP。

硬 IP 是指表示为掩码布局的设计；固 IP 是指特定技术的合成网表；软 IP 是指具有可扩展性和内置参数的 HDL 版本的核。后者的术语演变为“可参数化 IP”。它们可以进行设计，因而可以与硬件合成为一系列的规格和工艺。用于 DSP 应用的参数，如滤波器抽头大小、变换字长点的大小（Erdogan 等　2003，Guo 等　2004，Hunter 1999，McCanny 等　1996）可以制成为一个可编程功能。参数控制这些特征的合成过程中会被送入编码，从而得到该应用程序所需的硬件。每个类型的 IP 核都有优点和缺点，如图 10-1 所示。

这些类型的参数化软 IP 内核允许最大的灵活性，并提供复用的最佳水平，但在测试和验证方面会产生主要成本，因为它们在每一个模式都需要测试。此外，有一种推测认为，内核将在不同的 FPGA 技术下工作得最好，或者在当前的 FPGA 技术下改变参数，会使性能呈线性增长。这些问题会降低设计的可靠性。专用的功能可以预见的硬 IP 核，例如 Xilinx FPGA 的 PowerPC，就不会出现这些问题。然而，这里的主要限制是，这个平台不适合 DSP 应用程序，由 FPGA 提供的并发性是把选择 FPGA 平台放在首位的主要原因。因此，灵活性受到了基础专用硬件平台的极大限制。

虽然固 IP 和硬 IP 设备的设计是固定的，但它们仍然具有一定的灵活性。在

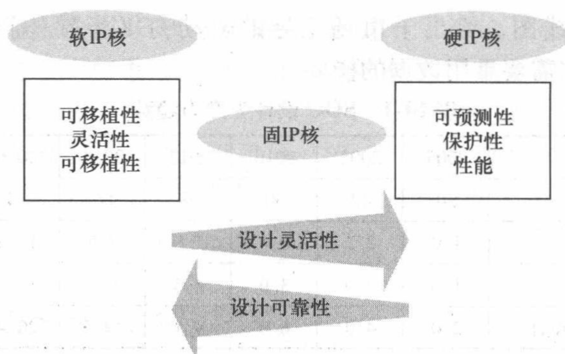


图 10-1 IP 核的优点

这些情况下，定义核心的 IP 参数被称为静态 IP（Junchao 等 2001），借此最终设计的寄存器内部可以设置为允许内部电路的复用，以便重新配置设计功能。重构已经成为 FPGA 的热点课题，尤其是它们正处于性能提高期（Alaraje 和 DeGroat 2005，Sekanina 2003）。与之相对的，软 IP 内核的 IP 参数被称为动态 IP 参数。他们往往是局部或全局参数，如数据宽度、存储容量和时序延迟。控制电路也可以被参数化，使设计具有可扩展性。参数也可以被设置为允许相同的主代码，在从 ASIC 库到不同 FPGA 应用的各种目标技术中实现优化。

许多公司提供 IP 产品是基于 DSP 解决方案的，即把 IP 代码嵌入到 DSP 处理器。这提供了充分的灵活性，但在分区、功率和速度上的性能被明显削弱。TI 公司和 ARM 是非常成功的由公司同时提供的芯片组和支持嵌入式组件库的两个例子。以类似的方式提供固 IP 核和软 IP 核的公司已大量建立。这使 FPGA 公司不仅出售实现用户设计的芯片，也提供创建这些设计的基本建设模块。这种可变函数库的可用性和 FPGA 的空白区，为即使是最小的设计团队带也来了巨大的力量。他们不再需要依赖特定领域内的专家，这使他们专注于整体设计，同时可确信 FPGA 供应商提供的核心已通过先前公司的使用测试。

这里列出一些当前的 IP 供应商（Davis 2006）如下：

4i2i 通信有限公司：FPGA 和 ASIC 中的 Verilog 和 VHDL IP 核。专业视频编码技术和 DSP 技术（http：//www. 4i2i. com/）。

科胜讯，安菲西恩 IP 核：FPGA 和 ASIC 中的 Verilog 和 VHDL IP 核。该公司专业从事视频、成像、安全、语音和音频、无线宽带和 DSP 技术（http://www. conexant. com/products）。

Axeon 有限公司：Vindex IP 内核——可综合、扩展的微处理器架构，经过优化可以支持机器学习（http://www. axeon. com）。

ARC：可配置内核：CPU/ DSP（http://www. arc. com/）。

数字内核设计：微控制器、总线接口和运算协处理器的 VHDL 和 Verilog 内核（http://www. dcd. com. pl/）。

这表明了 IP 产品的多样性，以及组件规模的加大。大核趋向于在专用标准指标有性能需求或性能受限的领域，如在有视频编码（JPEG，MPEG）及总线和存储器接口的情况下。

硬 IP 核内部组件可以被进一步定义（Chiang 等　2001），尽管这些定义可以被 IP 内核的所有变化应用，硅前阶段涉及更多的软 IP 核，生产阶段涉及预实现固定设计硬 IP 核。

投产前：如果设计通过仿真验证，则给予一星评级。

铸造验证：如果被特定过程证实，则给予三星评级。

生产：如果核心被证明可生产，则给予五星评级。

在开发 IP 时，厂商往往提供低成本的交易，以吸引系统设计人员能够使用他们的新产品，并验证它的成功。一旦硅芯片被证实成功，成为有市场优势的产品，就成为了具有竞争力的产品。

10.3　IP 核的演变

随着技术的进步，芯构件粒度的复杂性增加，设计抽象层次得到提高。这导致了基本构建模块层次结构的设计演变。本节将给出这种演变的总结。

在芯片领域，数据库的发展带来了高级别的粒度合成。库在最低级别定义的门控功能和寄存器。随着粒度增加，数据库可实现如 UART、以太网、USB 控制器等功能的合格功能模块。与此同时在 DSP 处理器领域，如 TI 公司这样的企业，已经成功地为自己设备上的应用生产软件解决方案。随着运算功能的发展，IP 核在 FPGA 和 ASIC 和重用设计中发挥了作用。这是支持 ASIC 合成基本构建模块的自然进展。致力于研究用复杂有效的方法执行一些最基本的算术运算的价值，有助于产生性能标准极具吸引力的高度复杂的 IP 核的设计。

图 10-2 所示为 IP 内核的这种演变，以及它们如何用更低级的模块形成关键部件增加复杂性，以达到更高级别的抽象图示。算术组件模块展示了大量关键的数学运算，如加法、乘法和除法，解决了如超前进位和高基运算等的技术问题。第 3 章涵盖了很多达到这一功能级别的技术。

随着较复杂芯片的到来，所述运算组件成为复杂分层结构的下一级构建模块，例如滤波器组所组成的大阵列 MAC 模块。这导致了如 FFT 和 DCT 的基本 DSP 功能的发展。这些实例以矩阵为基础运行，其中包括了大量在软件中表现欠佳的重复计算。它们可以用许多基于乘法和累加操作的关键构建模块来建立。算法的结构化性质使它具有可扩展性，允许使用很少量的参数来控制设计出的体系

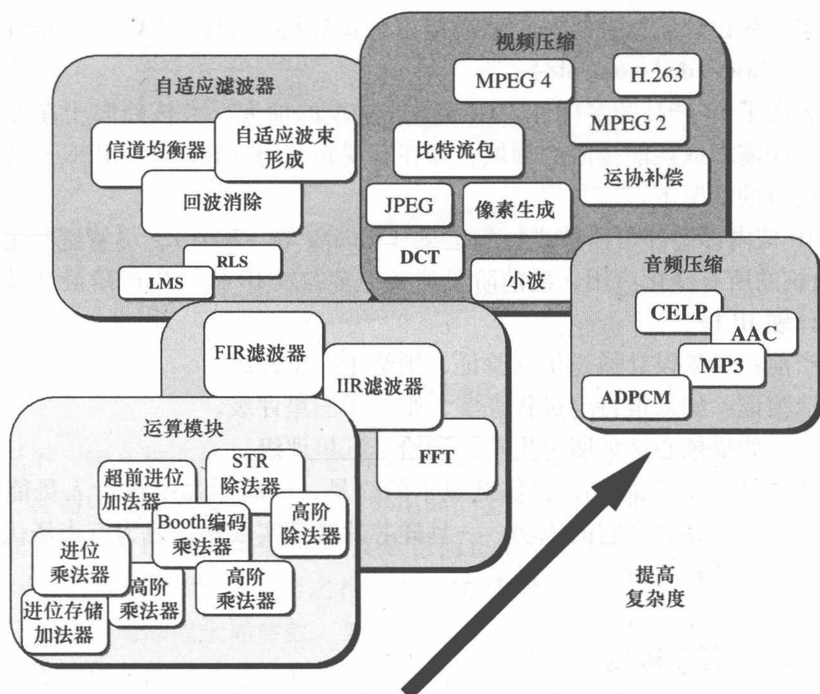


图 10-2 IP 核的演变

结构。例如字长和截断的选择。还有基于该矩阵运算规模的其他例子，比如与过滤器上的抽头数有关。这使得面向单一应用的工作可以被扩展以满足各种应用的需要。

其他更复杂的基础模块是从基本的算术功能开发而来的。以滤波器为基础的更复杂的例子有通过 LMS 算法或更复杂的基于 RLS 算法的 QR 实现自适应滤波器。后者在第 12 章的高等数学运算中给出了一个很好的 IP 核设计的例子。其他的例子，如前向纠错链和加密，因为有一个高度复杂的数值处理，所以也已经非常成功。

当前 IP 内核的成熟水平，已经能够完成先前需要大量独立逻辑模块或设备的全部功能。这再一次提升了基础逻辑块的级别。DCT 就是这样一个例子。DCT 的高性能 IP 模块的发展已是被热切关注的领域（Hunter 1999）。其中的一个复杂组件正是 JPEG 和 MPEG 图像压缩算法的基础，其本身就是商业化 IP 产品。

下面的章节将讲解每一层抽象设计所涵盖的更多细节。

10.3.1 运算库

图 10-2 所示为一些基本的数学运算，即加法、乘法、除法和二次方根。即

使是最基础的加法的高效硬件实现，也推动了一个领域的研究，即把操作分解为最低位水平的抽象，并巧妙地进行这些操作，在分区、时钟速度和输出时延方面提高整体性能（Ercegovac 和 Lang 1987，Hwang 1979，Koren 1993，Schwarz 和 Flynn 1993，Srinivas 和 Parhi 1992，Takagi 等 1985）。以下部分将介绍有关运算组件的选择及如何在代码中加入参数的一些细节。

1. 定点和浮点运算

算术运算可以执行使用定点或浮点运算。如在第 3 章所讨论的，位宽被定点运算分成一个定宽幅度分量和一个定宽小数部分。由于位宽固定，因此为了确保所得到的值是准确的，上溢和下溢检测则至关重要。前面的图 3-2 所示的数字浮点运算提供了更好的动态范围。

虽然定点设计和浮点设计的数据宽度内的记数法不同，但是操作的主要功能有所重叠，如图 10-3 所示的乘法示例。目前已有关于从定点自动转化为浮点的研究（Shi 和 Brodersen 2003）。

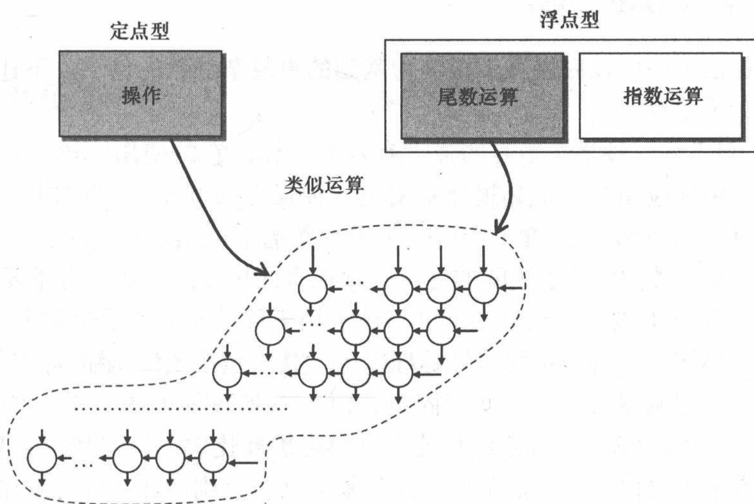


图 10-3　定点型和浮点型的运算

正如第 3 章讨论的，浮点运算和定点运算各有其优点和缺点。莱特博迪等人（2007）实现了让 FPGA 上的 QR 滤波器使用浮点运算。早期的 QR 滤波器研究（Walke 1997）显示，相比定点运算，浮点运算是更有效的特殊应用（雷达自适应波束形成）解决方案。然而，这次调查集中在 ASIC 应用上。由于 FPGA 的存在，定点运算在与浮点运算抗衡的路上从未见天日，似乎只能依赖于未来的 FPGA 器件的功能。事实上，FPGA 若缺乏专门的浮点硬件，则工作性能会很差，Craven 和 Athanas（2007）的研究表明，其性能可能比处理器更差。

2. 加法、乘法、除法和二次方根

高性能运算组件的应用需要大量广泛的工作（Ercegovac 和 Lang 1987, Hwang 1979, Koren 1993, Schwarz 和 Flynn 1993, Srinivas 和 Parhi 1992, Takagi 等 1985），其中一些已经在第 3 章中强调了。对于加法和减法还有乘法，专用定点加法和乘法硬件的演进否定了进行关于应用详细讨论的需要，在很大程度上，除了浮点，专用核心是无关紧要的。奥德菲尔半导体（现科胜讯）和东北大学开发了 FPGA 和 ASIC 的浮点运算核。第 3 章强调了数种除法的执行方式，包括循环法和功能迭代法。这些技术可以用明确的乘性和加性阶段进行架构描述，以扩展到满足字长要求。平衡的关键是使用 LUT 创建初始估计来加快这一进程，但这使得核心的扩展性变差。

随着 LSI 系统设计的抽象层次超越了位级别的算术计算描述，像 Synplicity 这样的工具使关键的定点乘法和加法运算部件可以使用简单的算术操作数。FPGA 设备已经可以实现高性能定点乘法操作。

10.3.2 基本 DSP 功能

本节提供了一些以底层算术模块为基础的更复杂的核的例子。下述算法的更多详情参见第 2 章。

FFT：FFT 是一种基于矩阵的强大有效的运算。它的使用广泛，可以被用在正交频分复用等应用中。正交频分复用是一种强大的调制/解调方案，用于如无线（IEEE802.11a/g）或广播（DVB-T/H）等通信应用。

DCT：这是另一种基于矩阵的运算，在很多图像处理应用中有重要作用。

LMS 和 RLS 滤波器：自适应滤波算法被用于移动通信和雷达等多种应用。

微波：小波分解有一系列信号处理应用，其中包括图像压缩。

滤波器：滤波器架构可以高度固定，由一系列收缩的重复单元组成，例如 MAC 操作。显然，这样的固定结构能够创建可扩展代码，以支持一系列滤波器。如果硬件水平需要保持在最低限度，则其复杂度会急剧增加，硬件重用专注于核心业务调度算法的完整功能。

在所有情况下，以这种方式获得的电路结构，是为了保持规律性和可编程性，因为这将允许核参数化，并跨参数范围产生相当一致的性能结果。拿 FFT 来说，FFT 可以这样分解，即用更小的模块来创建更大的 FFT。很明显模块大小可作为参数，尽管它可能不是最有效的面积和速度的解决方案。然而，模块大小参数会带来良好的性能。一个可扩展加法器的可用性也意味着对于不同的字长，性能是可预测的，但由于 Xilinx 公司的 Virtex（18 位）和 Altera 的 Stratix（36 位）的 FPGA 族限定字长，因此这成了乘法器功能的难点。一旦超过这些字长度，则乘法器不得不被限制在数个模块中。

10.3.3　复杂的 DSP 功能

上面所总结的 DSP 功能形成应用范围内的主要算法功能，并需要通过关键参数进行扩展。然而，也有一些应用的性能有非常好的标准定义，例如 JPEG 和 MPEG。如所讨论的，DCT 形成 JPEG 压缩中的一个关键功能。如图 10-4 所示。一个关键的平衡是直接在硬件上创建更复杂的 DCT 功能，并把功能的其余部分制成软件。那么如有需要，就可重用硬件模块来制造更大的模块。MPEG4 视频编码涉及占用 90% 处理时间的额外动态估计和补偿模块。因此，存在在用于动态评计的参数化内核。

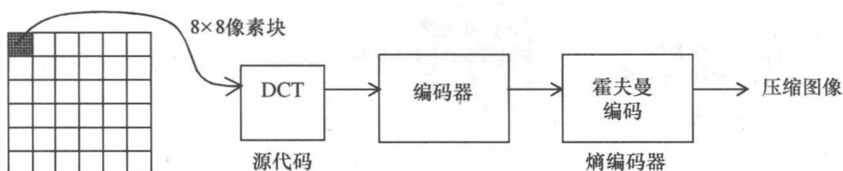


图 10-4　JPEG 图像压缩

10.3.4　IP 核的未来

随着核构建模块中设计抽象层次的提高，设计师的作用是系统集成，特别是利用现有的 FPGA 器件在一台设备上实现完整的系统功能。由于 IP 核的使用继续增长，因此设计流内的其他方面亟待处理。以下引用自 Gajski 等人（2000）：

“IP 工具开发人员为 IP 供应商和系统集成提供设计方法和工具，以支持 IP 开发和系统集成”。

以下部分将详细介绍从设计理念到可扩展的（软 IP）解决方案的过程。

10.4　可参数化（软）IP 核

本节将涵盖实现 DSP 功能的参数化 IP 核的开发。一个数学组件的硬件设计的起点可以是来自该算法的 SFG 描述。这里给出的算法图示展示了设计所需组件内部的相互依赖。该描述可以处于从位级算术运算到小区级别功能的不同层次。SFG 通过这些组件给出数据流的相关信息，并从可开发硬件体系结构中提供一个非常强大的设计描述。

图 10-5 所示 SFG 的例子描绘了执行复合乘法的两种变化。这个简单的例子说明了通过重新排序算术运算，作何更改可以增强区域或所得的硬件体系结构的关键路径。$(a+jb)$ 和 $(c+jd)$ 两个数字的复数乘法的解决方案为：

$$(a+jb)(c+jd)=[a(c-d)+d(a-b)]+j[b(c+d)+d(a-b)]$$

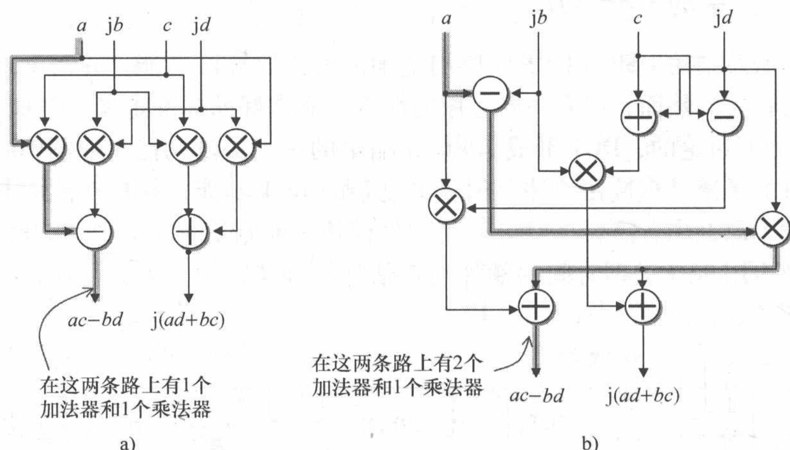


图 10-5　复合乘法器 SFG

a）4 个乘法器/2 个加法器　b）3 个乘法器/5 个加法器

图 10-6 所示为从算法的 SFG 表示开始，基于 DSP 的电路设计的传统设计流程。通常情况下，由于设计变化或未完成预期成就，此循环将重新迭代。许多步骤在前面的章节有提及。该过程开始于用于 SFG 或 DFG 描述的算法定义。然后进行在第 2 章中强调的细节分析，确定如内部字长、截断、吞吐率问题等的关键方面。第 8 章详述了创建电路结构的界限。电路结构被编码成一个基于 HDL 描述，并创建最终设计的传统集成工具。如果某些方面规格需要改变，如字长度，则传统的全设计流程需要全部重复。

参数化 HDL 的 IP 核发展使得该设计流显著改变，如图 10-7 所示。设计的初始研究需要研究字长和算术对 SNR、面积和时序性能的影响。要着力确保包含额外的流水线时其运转仍然是准确的，目的是使核的参数无缝地导致靶向范围的规格精确实现的库，而无需更改代码的内部工作。系统应有效地允许大量参数被反馈到代码的顶层。它将通过不同抽象层次的代码传递到最底层。显然，在架构层面需要相当大的努力来开发这个参数化的电路架构。在时间和精力方面的初始成本无疑阻碍了扩大设计的再利用原则的使用。但是，在该初始经费上，公司可以做到节省大量的时间和金钱资源。选择什么设计组件，基于什么 IP 进一步设计和发展，是这一成功的关键。初始成本必须从长远来节省资源。

未来的设计工程师需要学会如何对一个完整的设计实现从头到尾的重用方法。在设计过程中需要考虑的问题，例如字长影响、硬件映射、时延和其他电路 HDL 模型可以生成之前的时序问题。需要考虑的方面创建到设计过程中一个全

新的层面上，和设计师需要保持在任何他们生产的无论是开发或测试目的头脑可重用性。如果一个设计以参数化的方式开发，那么最初的分析阶段可以从设计流程来消除，如图 10-7 所示，使附加电路在极短的时间内开发和布局规划，通常以天而不是月为尺度。这代表了一个 IP 核开发人员（Howes 1998）的明确市场，因为它可以大大加快客户的设计流程。然而，IP 核公司可参数化开发需要一种不同的设计方法，它将为一系列应用提供有质量的解决方案。

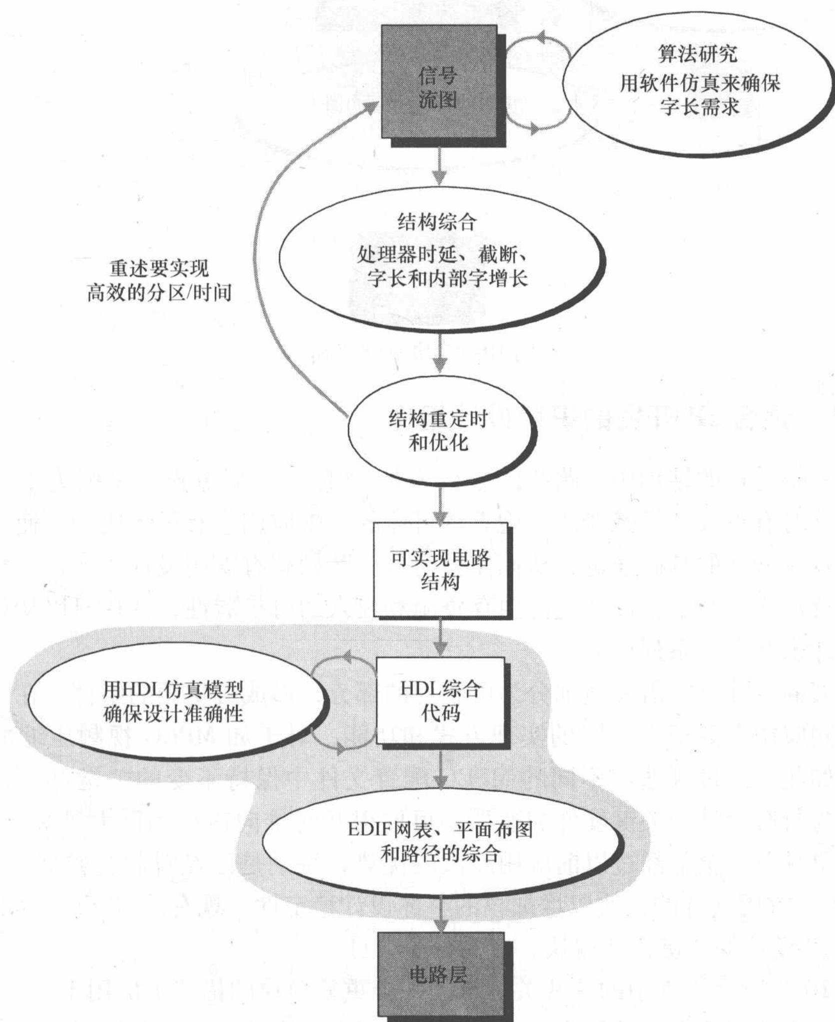


图 10-6 VLSI 电路设计流

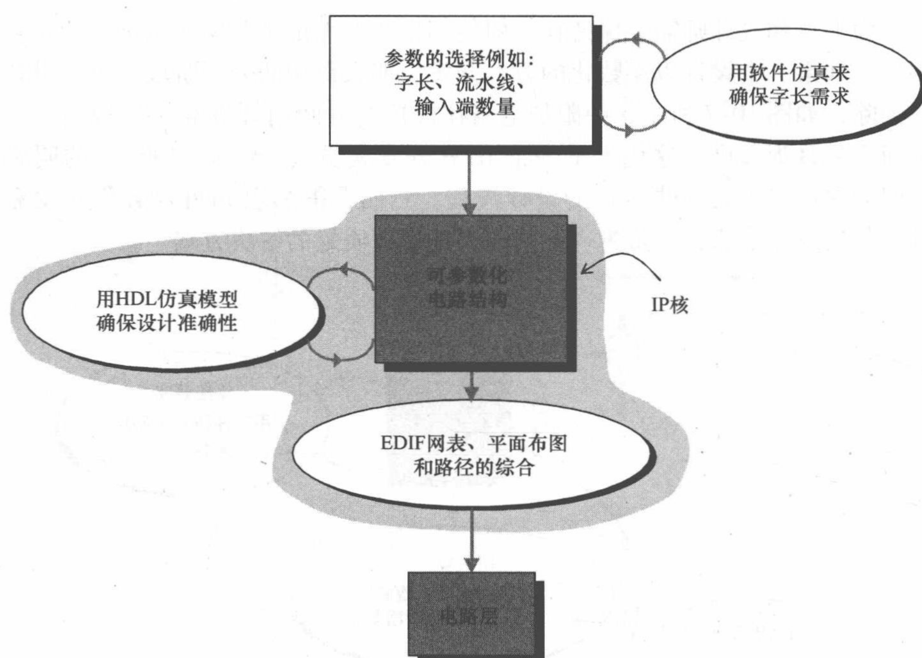


图 10-7 快速设计流

10.4.1 适合 IP 开发的识别设计组件

在一个公司的结构中，路线图被认为是 IP 库的发展重点，是因为重用概念引入设计时有更大一笔经费。客观看待可能的未来应用会有更大成功，使开发流水线可以从最初的基础演进，从而使公司从一开始就有重用设计。通过参数化的字长和流水线的水平，以及允许内存资源和投入的可扩展性，往往可以从相同的种子设计来开发一系列产品。

较大的设计可能需要被细分为可管理的部分，形成可重用的组件。在一定范围的不同应用需要稍有不同的实现方式和功能，对于如 MPEG 视频压缩的大设计更是如此。通过挑选出不同的 MPEG 配置文件中保持不变的关键部件，使用它们作为所有设计的关键硬件加速器，可取得及时推向市场的巨大进步。此外，现有模块具有已全部经过以前应用测试的优势，特别是已经对制造或部署到 FPGA 上的。重用这样的模块以增加核的整体设计稳定性。现有的 IP 也可以形成更高层次 IP 设计的关键构建模块，成为分层设计。

图 10-8 所示为其中的一些关键点。一个重复模块的描述被应用于多个设计。类似地，图中表示了用低级模块分层级建立更大的设计。公司路线图突出了仔细选择用来指派额外设计组件的重要性。可扩展性是另一个重要的考虑因素，即设计可否被扩展，并调整为满足各种应用的要求。语音压缩是这样的一个例子，编

解码器可能需要工作的信道数目范围取决于整体应用，例如，是一个简单的 DECT 电话还是电信网络。

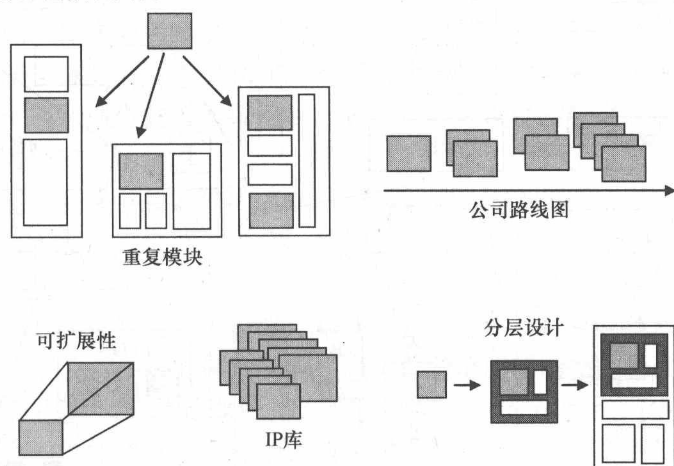


图 10-8　IP 的关键构建模块

10.4.2　确定 IP 核参数

识别 IP 核开发中的关键参数，需要对该芯的使用范围有详细的了解。这样做的目的不是仅仅创造尽可能多的灵活性，也为了确认这些额外工作将带来的长远利益。过度参数化设计不仅影响开发时间，而且还影响到需要考虑所有核排列的验证和测试。换言之，可以在考虑设计时间和设计性能时增加一个额外的可变的影响，并权衡增加的灵活性将如何拓宽 IP 核的使用范围。

以下是示例参数，其中一些如图 10-9 所示：

1）模块/架构；

2）字长；

3）内存；

4）流水线；

5）控制电路；

6）测试环境。

一个明显的参数是字长。字长的选择涉及信噪比和性能标准之间的协调，如区域和关键路径。图 10-10 所示为不同字长的信噪比。从这个简单的例子中可以看出，字长的进一步增加不再显著提高整体性能。对于某些组件，如加法器，增加一位将线性扩展应用结果的面积。然而，对于乘法器和除法之类的组件，增加字长将对区域产生巨大影响。因此，字长分析和截断或吞吐分析一样重要。

实现硬件的完全扩展性是另一个例子。该图展示了通过单一模块的重复满足

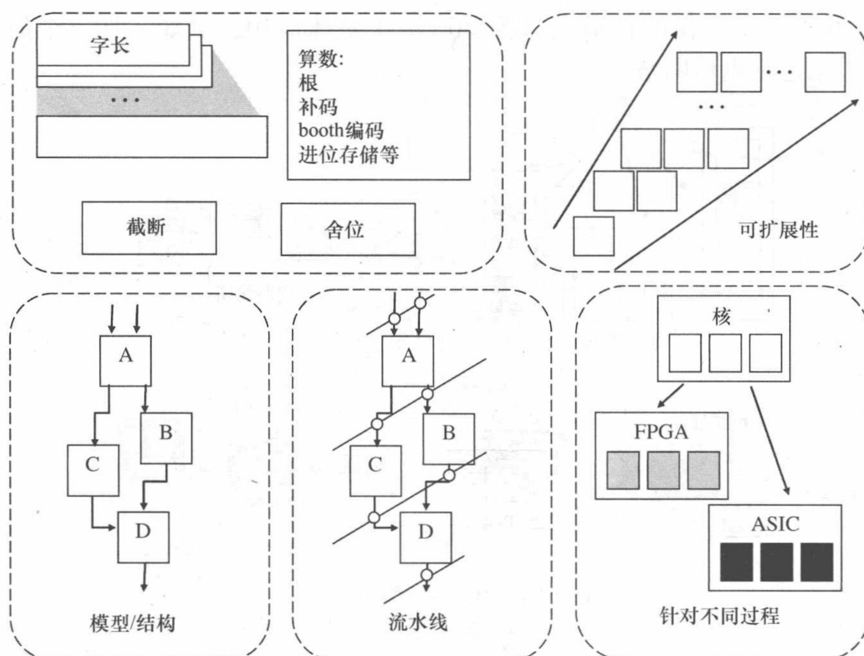


图 10-9 IP 参数

应用所需要的规模。N 抽头滤波器就是这样的例子。随着 N 的增大，硬件应该相应地增加逻辑块，以满足应用要求。

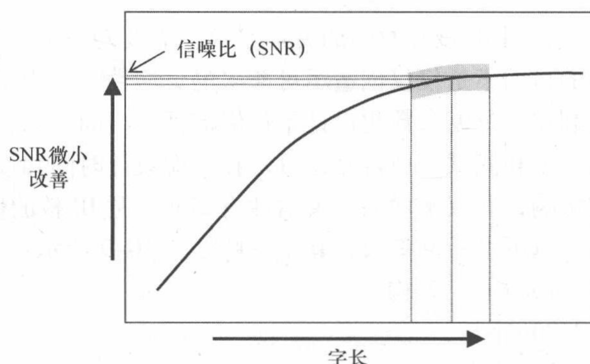


图 10-10 字长分析

由于不同的字长及输入或存储值的数量变化，存储器也需要被扩展。更重要的是如何创建存储器架构。第 5 章的表 5-3 和表 5-8 给出了可用在 Altera 的 Stratix 和 Xilinx 公司的 Virtex FPGA 系列的一系列内存。很显然，嵌入式 RAM 为面积提供了高效粗存储模块，但嵌入式 LUT 由于其操作的高度并行性且被共置于硬件，因此可允许更快的操作。在核的开发中这也需要被考虑到。

组合子模块可以实现不同的应用。在图 10-9 中，它们被标记为 A，B，C，D 模块，描述了一个简单的架构。区域内的字长度的附加位将对关键路径具有连锁效应，从而可能导致需要在使用 A，B，C，D 各子模块的相同简单结构的设计中引入更长的流水线阶段，如图所示，允许流水线的层次不同是可拓宽应用机会的一个重要的参数化特征。

应用的数据速率的灵活性和相关时钟速率的性能也将需要给设计中的流水线分层，以满足关键路径的需求。如已经提到的，这可以配合字长度的变化和流水线数目的经常性变化成为可重复使用的算术核心的一部分。显然，一个模块内流水线数目的增减，将对关联模块和和更高层结构的时间和调度产生连锁影响。由于这个原因，低层组件内的流水线控制，必须从模块设计的较高层做，从而使较低层的代码不再需要手动编辑。

如果参数发生变化，如额外的流水线时延，输入信号或字长数目的增加等，则有必要开发可扩展控制电路。也这可能对模块的周期和时序产生影响。一个重要的例子就是如何在单一的硬件上开发可参数化控制来处理多个操作进程。

一个完全参数化设计是一个需要计划的难题。总体目标是，代码可以被在极短的时间内再合成为一个新的体系结构，同时仍满足所需的性能标准。这是一个 IP 核成功的关键因素。至关重要的是，所得到的核具有在面积、功耗和速度方面媲美手工设计的性能数据。同样的，该过程涉及时间、金钱资源及核的性能标准之间的平衡。

开发可参数化核的另一种方法可以是创造软件代码来自动化 HDL 模块的脚本。此方法在使用 Verilog 时特别有用，因为在可扩展设计中，它不具备 VHDL 的灵活性。

设计中的参数化水平也必须考虑到。从一开始就让设计拥有尽可能多的灵活性似乎是合理的。这样做必然会拓宽 IP 核的市场潜力，突出过度参数化的问题。Gajski 等人（2000）指出了由于参过度参数化的问题：核内的变量越多，越难验证设计的每个排列的全部功能。另一方面，已经过于通用化的设计可能无法满足特定应用的性能要求。Gajski 表示，参数数量的增加，降低了满足用户需求的设计质量和特点，也增加了验证和测试的困难。图 10-11 突出了这些问题，有必要在选择参数时考虑这些问题，以找到设计灵活性和可靠性之间的平衡。

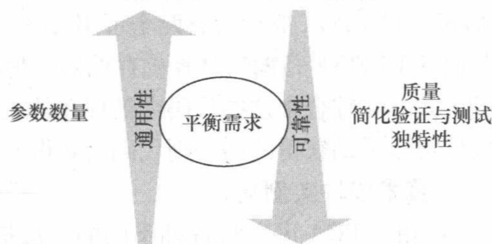


图 10-11　基于设计可靠性的综合的影响

10.4.3　针对 FPGA 技术的参数化特性的发展

许多针对 ASIC 应用的 IP 设计可被扩展为 FPGA 应用，反之亦然。每种技术都有其自身的特点，但由于这些差异，在选择 FPGA 或 ASIC 族时在顶层重新面向核的代码变的可行。这在 FPGA 被快速处理中特别重要，因此遗留代码需要为未来的器件和封装提供空间。图 10-9 所示为如何把一个种子设计用作面向 ASIC 和 FPGA 的具体技术设计的基础。

同一代码的两种技术之间有明显的代码重用优势，通过不限制目标的技术扩大产品市场。然而，它也适用于验证框架，即以 FPGA 为原型的核；相同的代码会被重靶向到 ASIC。从 FPGA 到 ASIC 应用的代码转换实施本身显然无法保证不出错。但是，验证实时 FPGA 平台上的代码功能为设计过程带来了极大的稳定性，并使得功能设计得到加强，以更好地满足规格需要。在两个定向应用的主要流之间切换时通常需要下列问题：

1）FPGA 嵌入式运算；

2）ASIC 定制设计的运算核；

3）特定技术内存。

流水线需要在技术、区域和资源的选择上被调整为满足所需的性能。与 ASIC 相比，FPGA 资源有限。因为成本原因，设计师不妨保持 FPGA 器件的特定档次和规模。限制流水线的数量会影响可达到的最大时钟速率。

FPGA 应用的 IP 组件极大地加快了在单一设备上的全系统开发。此外，像 Xilinx 和 Altera 这样的供应商已经与外部设计公司合作开发面向他们的设备的函数库，并可在官方网站上下载，特别是 Xilinx Alliance 项目和 Altera 的 Megafunctions Partners 项目。获取这样的材料提高了用户使用 FPGA 设备和拓宽市场的能力。这象征着 IP 核供应商、FPGA 公司和设计公司的共荣关系。

1. 内存模块实例化

FPGA 器件之间的区别之一是存储器模块。每一族模块都有其自己的架构。使用 Verilog 的 FPGA 应用可有两个可用方案。目标设备的 RAM 模块实例化可以用顶层代码的 DEFINE 指向所选择的存储器。另外，代码可以被写成如存储器应用中涉及的方式，在综合中这些代码会被现在的工具实例化为存储器。但是，如果存储器的实例可以手动获得的话，则代码会获得一些改进，但也会将更加复杂。

2. 算术模块实例化

不同的 FPGA 可以将各种不同的算法提供给用户使用，使这些嵌入式模块可以在代码中进行推断。然而，建立算法需要定制更大的位宽。另一个方法是将这些嵌入式模块用作核运算模块，来实现浮点型算法区。如果这些代码需要在不同的 FPGA 之间，甚至是 FPGA 和 ASIC 之间使用时，就需要一个设备在顶层定义这种算法。在 Verilog 中，这将通过使用顶层文件中的 DEFINE 来允许用户自定义设计，从而达到当前的需要，并为他们提供三种选择：

1）推断算法：如 $A+B$ 等；

2）FPGA 内置运算实例化；

3）自定义算术运算实例化。

后者可能是大数值的乘法或浮点型算法的更好选择。如果为满足关键路径的使用要求需要额外的流水线，则使用这些自定义算法也是很重要的。此设计依赖

性也需要进行分析。

3. 参数化设计和测试环境

IP 核所有相关测试代码的设计都应有可扩展性的思想。用于功能验证的精确软件模型应具有不同位宽来匹配 IP 核。为周期精确测试，也必须考虑到时延。测试平台和测试数据推导也需要可参数化。所以允许使用一个完全自动化的代理 IP 核及其相关联的测试软件。在 IP 核开发中，使用如 C 语言软件生成的测试工具和测试数据是很有优势的，如图 10-12 所示。

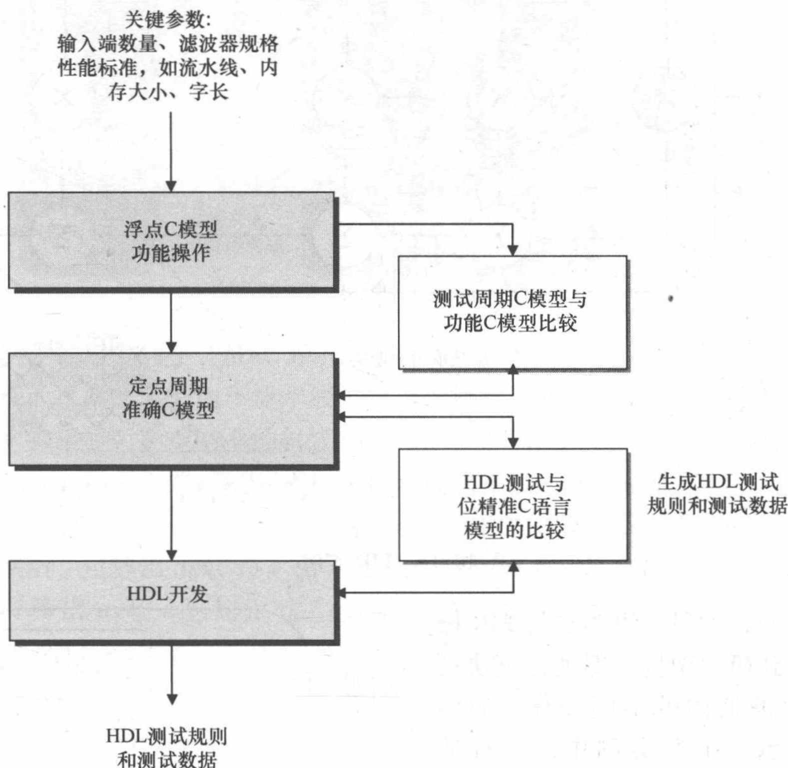


图 10-12　测试设计流

10.4.4　简单的 FIR 滤波器应用

本节以适用于图 2-13 的一个简单 FIR 滤波器的参数化设计为例，设计的关键参数将被明示，并给出如何把它们编写进代码的提示。更完整的 FIR 设计实例在第 8 章给出。

本实例的 FIR 滤波器有 4 级层次结构，单独的运算单元为最低级，全 FIR 滤波器作为最高级：

1 级：这是输入 $x(n)$ 和输 $y(n)$ 过滤器的最顶层。

2 级：FIR 滤波器的顶层可用一个简单的 $a_0x(n)$ 乘法器构成，随后是多个 MAC 时延模块，如图 10-13 中的阴影框所示。

3 级：这是算法运算层，由乘法、加法和时延模块组成。

4 级：运算模块可细分到更低层，执行的位级运算通常与 FPGA 不相关，除非设计者用加法器和 LUT 构建乘法器。

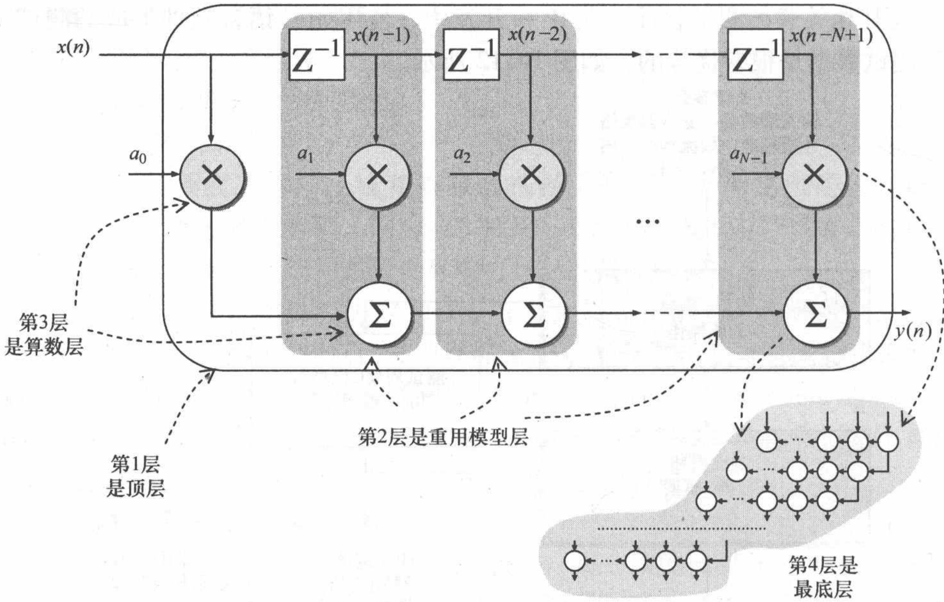


图 10-13 FIR 实例

这种设计的另一思路是将 FIR 操作的叠放到简化架构，即硬件模块被重用于过滤器内的不同操作，如图 10-14 所示。在本实例中，所有的 MAC 运算在一组乘法器和加法器模块中执行。多路转换器用于控制来自 MAC 运算的输出和返回算术模块的数据流。操作单个单元的技术问题超出本章的范围，详见第 8、9 章。

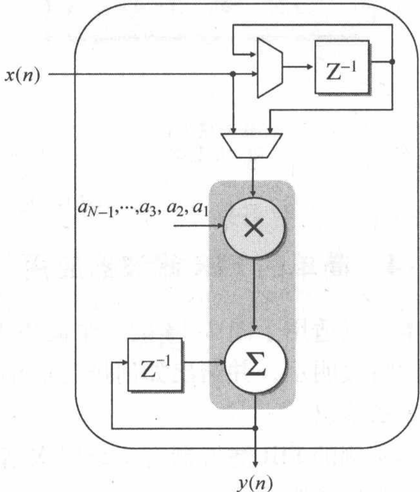


图 10-14 缩减硬件的 FIR 例子

对硬件简化程度的选择取决于应用的性能需求。对于同一实例，我们可以提供数据来对未经硬件简化的例子和单一 MAC 单元的例子作性能比较。图 10-15 所示为 6 级 FIR 的例子，

公式为

$$y(n)=a_0x(n)+a_1x(n-1)+a_2x(n-2)+a_3x(n-3)+a_4x(n-4)+a_5x(n-5)$$

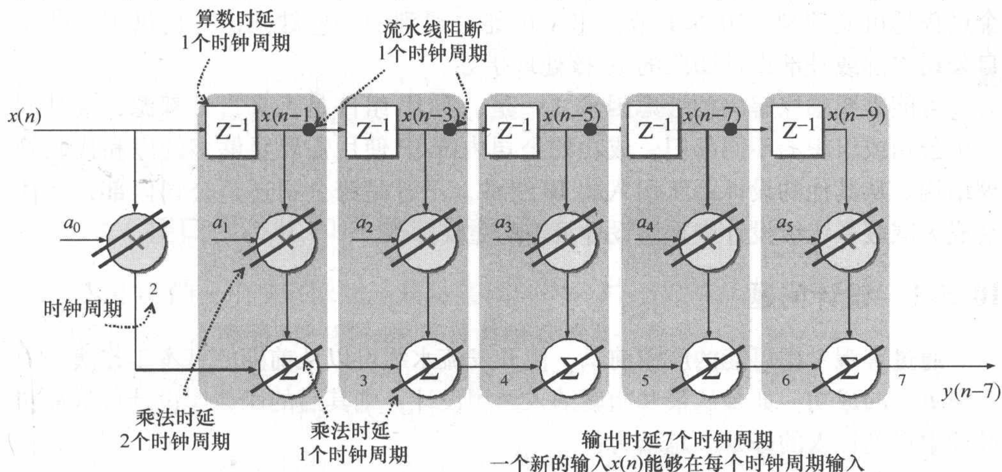


图 10-15　6 级 FIR 时延实例

这个例子的目的在于，MAC 单元内的乘法和加法模块的时延分别为 2 个和 1 个时钟周期。这些时延是由于流水线被置于模块内，并可用 FPGA 实现。运算模块内的流水线阻断由图 10-15 的斜线表示。由于算术模块内的时延，附加流水线切割被标记，并用于电路的时间重设和正确地调度模块中的运算。

该图中的数字表示已执行操作的时钟周期。例如，由于乘法器的时延，在两个时钟周期后，可算出 $a_0x(n)$。由于该算法时延加上乘法器的时延，三个时钟周期后，可算出 $a_1x(n-1)$。这些值需要进行整合。然而，匹配 $a_0x(n)$ 和 $a_1x(n-1)$的结果的调度，$a_0x(n)$需要再延迟一个时钟周期，这在第 8 章有详述。$y(n)$的输出结果总时延是 7 个时钟周期。换而言之，$y(n)$是在 $x(n)$输入到滤波器的 7 个时钟周期后输出，并在每一个后续时钟周期都有一个新的结果。再一次的，FIR 操作被几个 MUX 映射到一个 MAC 单元。这意味着所有的操作需要调度适当，使得它们在正确的时间执行正确的输入。算术单元的时延同理，这部分在第 8.6.2 节提到过。同样的，选择恰当的体系结构能带来性能需求、面积和时序标准的平衡。

10.5　IP 核集成

成功的设计重用的一个关键是用户系统设计内的 IP 核整合。这通常是开发中的难点。之前的调查（Chiang 等　2001，Gajski 等　2000，Moretti 2001）已经突出强调了这些问题，而另一些人已在试图规范这一过程（Birnbaum 2001，

Birnbaumand Sachs 1999，Coussy 等 2002，2006，2007）。

为了实现把 IP 核成功融入当前的设计项目，必须采用某些设计战略，使这个过程尽可能顺利。10.5.1 节突出了可能会遇到的一些缺陷，并提供了一些来自公司内部或外部设计团队的 IP 核处理指导。

可能需要加以解决的考虑因素之一是，源 IP 组件是否来自外部源，或是同一个公司或组织的不同部门。成功的公司内部 IP 使用需要足够多的库和代码管理结构。从其他的设计团队引入的 IP 组件，不管是跨公司还是公司内部，往往会成为减缓系统级设计中重用设计策略的主要障碍。

10.5.1 设计问题

通过客观考虑可能的应用前景，使开发流水线可以从前期的基本工作演进中获得更大的成功。如果从最开始就纳入重用设计，则其后的函数库设计可从最初设计中得到惊人的好处。总结为：

1）需要确定将有益于未来的发展的部分设计。

2）可能的应用前景有什么？

3）研究未来的产品路线图。

4）有从同一个的种子开发一系列产品的可能性吗？

5）较大设计如何被分为便于管理的可重用部分？

6）查找现有的粒度水平，即有任何先前的 IP 可以提供起始水平的开发吗？

1. 外部 IP

外源 IP 的限制因素之一是可信度不高。IP 可以被分为不同的等级，一颗星表示此核通过仿真验证，三颗星级表示通过技术模拟也就是门级仿真验证，最后，五颗星表示 IP 核通过了实施验证（Chiang 等 2001），可信度最高。

FPGA 供应商与 IP 设计公司合作，为自己设备上的应用提供函数库，这给用户带来了一定程度的可信度。对于 FPGA，核的可靠性方面不似 ASIC 那般关键，但仍很重要。在把 IP 集成到用户产品上时浪费的时间，对该项目的成功至关重要。

以下问题可以帮助确定 IP 供应商的可靠性：

1）先前应用中的核能否用于其他用户？

2）公司供应用户指南和数据文档吗？

3）是否为此核提供验证平台和足够的、合适的测试数据？

4）公司是否提供集成支持，这会导致额外的花销吗？

2. 内部 IP

对于那些在只有一个工作地点的小公司，共享和分发内部 IP 将会非常简单。然而，对于所处时区和物理距离跨度较大的大公司，这会变得极为困难。此外还

有工时资源的竞争。项目领导并不乐意他们的团队花时间去把遗留代码集成到另一组的开发上。时间资源需要在公司的层面上被认知并评估来支持这种合作。

公司介绍 IP 核的设计结构，并在顶层设计上给予指导是恰当的。规定一个 IP 核的标准格式是有价值的，并使其更易于集成。一旦核被验证可行，则有必要形成一个中央存储库，使公司完全访问 IP。大多数公司已经使用了一些代码管理方法来保护他们的产品。

10.5.2　接口标准化和质量控制指标

使用 IP 的一个主要限制因素是集成问题，另一个是开发人员可能对第三方 IP 缺乏信心。本节将对一些标准进行简单总结。

虚拟插槽接口联盟（Virtual Socket Interface Alliance，VSIA）是一个成立于 1996 年的标准组织，用来支持电子行业中重用 IP 核开发设计的标准化。这是一个联盟，代表 SoC 产业的一个截面。11 年之后，该组织停止运作。在这段时间中，VSIA 建立了 20 多个标准，并用收费很少甚至免费将规范和技术文件分发给成员。他们已经将标准和工作交给其他发展 IP 和电子产品的标准组织来处理，如 IEEE。

他们特别成功的项目是 VSIA IP 质量（Quality IP，QIP）指标（VSIA 2007）。此免费文档涵盖了集成第三方 IP 时的主要设计考虑。它对供应商质量提供了技术指标，同时还有软核、硬核和验证 IP 的指标。出版后的数年里，它被很好地建设起来起来并广泛地为开发者采用。

除了支持文档，QIP 还可以由一个 Excel 电子表格实现，用户插入他们考虑使用的产品和供应商的特定信息和参数。电子表格由许多单独的表组成，一个是对供应商的评价，一些是软 IP 集成，另一些是 IP 开发。下面列出了一些询问供应商的问题：

过程：

IP 的开发过程被定义和记录了吗？

是否有用户满意度评估，并且会一直继续吗？

验证：

需求是否改变了 IP 的定义、记录和后续工作的进程？

是否有详细的协调测试计划，此文件可以提供给用户吗？

此 IP 核是否在实际生产环境中被使用？

版本控制：

修订控制计划和相关方针是否完全文档化？

IP 的使用截止日期是否会提前通知（至少 6 个月）？

分布：

如果 IP 可交付成果的支持改变，用户是否会收到通知，用户会收到新的 IP 么？IP 的新特点或新版本可用吗？

一致性：

IP 和相关联的交付结果所提供的标准一致吗？

支持：

使用中有问题报告措施和相应的程序吗？

可以在购买 IP 之前先测试评估吗？

文档：

文档是否涵盖把 IP 成功集成到一个系统的所有必要方面？

供应商的可信度：

该公司已经存在了 5 年以上吗？

这里有超过 50 个员工吗？

可以提供客户参考吗？

这是 QIP 电子表格细节问题的一个子集。关于软 IP 集成指标的问题如下：

1）IP 集成是由开发团队之外的其他人员做的吗？

2）有培训吗？

3）它是 FPGA 或还是集成电路？

4）有足够的文档吗？

接口信息，实例化指导；相关技术区域和力量评估；验证方法；可交付成果清单。

5）集成：

构建环境：提供脚本吗？可以扫描插入脚本吗？可移植到其他设计工具吗？

这个列表还远远没有完成。QIP 问问题事无巨细。

软 IP 开发表对计划开发一个 IP 核是很有用的指南。它可以作为一个提供优质代码的清单，并代表这一章所讨论的大部分内容。这是一些最相关的评论：

1）是否所有的时延值都是作为参数传递，而不是硬件编码？

2）如位宽、寄存器地址等值能用参数代表吗？

3）信号命名标准有一致性吗？

4）Verilog："define" 引用保存于一个文件吗？

5）VHDL：时延常数是从一个包获得值吗？

6）综合语句功能覆盖率能达到 98% 吗？

7）回归测试有日志文件脚本吗？

QIP 文档是一个非常有用的工具，因此得到了广泛的应用。它一直在不断修订，如今 VSIA 已经把它交给了 IEEE，它仍会继续发展下去。

VSIA 有一个 IP 保护工作小组，目前参与到 IP 核加密标准的开发中，还有

现有的软、硬标签标准。商业实体 ChipEstimate（www. chipestimate. com）及 Design 和 Reuse（www. designreuse. com），受益于 VSIA 的 IP 传输规范的使用。

10.6　ADPCM IP 核的例子

ADPCM 是 ITU 定义的众多语音压缩算法中的一种。对于从数字无绳电话到电话网络的众多应用，这是一个受欢迎的语音压缩应用程序。它提供了把 64kbit/s 的压缩脉冲编码调制（Pulse Code Modulation，PCM）音频信道压缩为 40kbit/s，32kbit/s，24kbit/s 或 16kbit/s ADPCM 语音信道，反之亦然。32kbit/s 的标准操作速度提供了高质量的压缩语音，相比 PCM 没有明显的损失。表 10-2 给出了 PCM 和 ADPCM 的关键 ITU 标准的概述。

表 10-2　PCM 和 ADPCM 的关键 ITU 标准的概述

压缩标准	速率（kbit/s）	特点
PCM（G.711）	64	从 14 位或 13 位统一 PCM 到 8 位对数 PCM 的转换使用 A 或者 μ 编码法则。当采样频率为 8kHz 时，8 位代表采样数据速率从 96kbit/s 减到 64kbit/s。
ADPCM（G.726）	40，32，24，16	此标准给予 G.723 和 G.721 标准。增加了可只使用 2 位来编码不同信号的灵活性，因此数据速率为 16kbit/s。（16kbit/s 过载信道用 DCME 传送声音）。
ADPCM（G.727）	40，32，24，16	此标准提供了 64kbit/s PCM 信道到速率可变（40，32，24，16kbit/s）嵌入式 ADPCM 信道的转换，此标准被用于允许不同信号被一系列核心位和增强位来表示。核心位是转换的必要部分。转换过程中，增强位拥塞的出现可能会减少，这样一来核心位不会丢失。

由于可使用 ADPCM 的应用程序的变化，开发一个符合各种需求的设计是一个难题。图 10-16 所示为从 PCM 输入的 APDCM 基本结构压缩。APDCM 压缩问题的症结所在是使用以前的语音样本对当前的样本做自适应预测。一旦计算出预测样本要从当前实际样本中减去，则只有这个错误信号被量化和编码传播。然后在解码端，接受方执行相反的过程重新生成实际的语音样本。

这个技术依赖于语音提供了某种程度的伪平稳性的事实。也就是说，在大约 20ms 的短时间内，信号的统计数据基本未变。这允许自适应预测用于 ADPCM。其他 ADPCM 的 ITU（G.726）标准有不同的量化器表来执行 40，32，24 或 16kbit/s 编码，并类似地解码。或者，可以使用一个用于所有四个压缩率（ITUG.727）的量化表，用 16kbit/s，24kbit/s，32kbit/s 数据率量化表作为 40kbit/s 数据率表的子集。然而，此优化需要稍微降低一点信噪量化比。

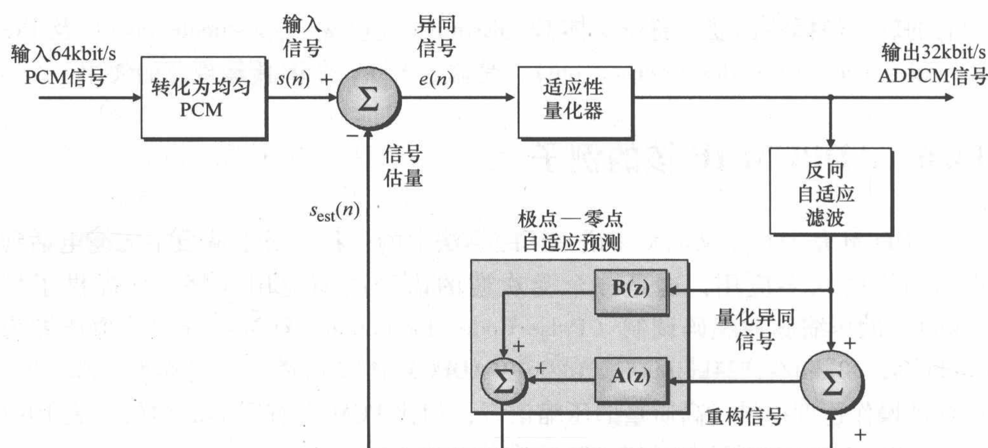


图 10-16　ADPCM 编码的基本结构

正如在数据编码（见图 10-16）和解码（见图 10-17）里看到的，某些组件成分一致，并可以用来执行这两个任务，因此有了开发具有双重功能的单一设备的机会。另外，独立模块可以开发编码和解码功能。通过扩展内存和寻址逻辑，或实现多个编码器和译码器模块实例化，可以实现可扩展性。

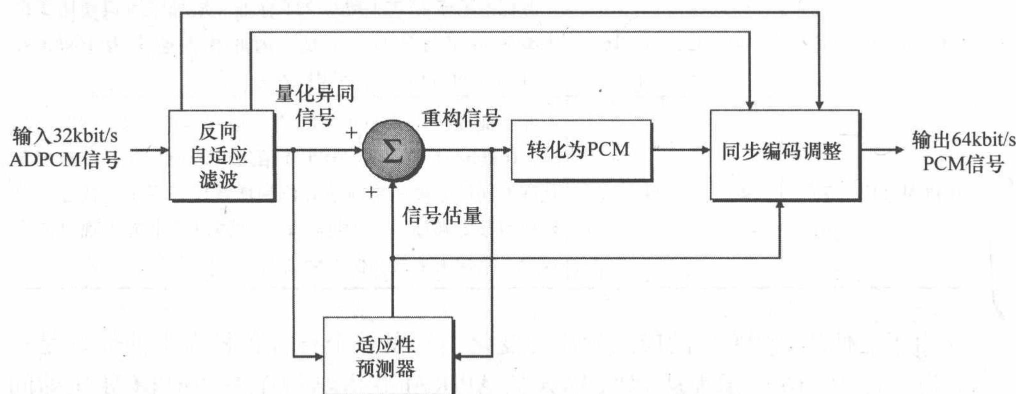


图 10-17　64kbit/s 自适应差分式 PCM 解码

ADPCM 的 IP 核应用考虑如下：

1. 所支持应用范围

从简单的 DECT 手机到如 OC-192 的高端电信系统，应用范围已经扩大到支持超过 1000 个语音频道。多频道核计算模块的使用需要纳入可扩展性（见图 10-18）。同样，多个核心计算块的实例化可能需要来满足积极的信道要求。请注意语音已设置对数据率和可交付的采样率的要求。只有两样都符合才可以使用。添加额外信道需要关键硬件可以管理计算，满足实时语音的需求。

2. ADPCM 标准的变化

ACPCM 和 PCM 标准有很多变体，其中一些在表 10-2 列出，并如图 10-9 所示。支持这些标准的广泛变化是明智的，因为这将增加普遍适应性。例如，PCM 有两个变体，被称作 A 律和 μ 律。A 律是欧洲标准，μ 律是美国标准。

3. 支持一系列 FPGA 器件和/或 ASIC 技术

通过加入代码或参数，同一个核设计可重新面向不同的技术，这使设计在面向 ASIC 之前先在 FPGA 上做出原型。这同样允许了不保障 ASIC 设计开销的低产量实现。

同样，这对新兴 ASIC 铸造厂的快速再发展能力极为有益。

4. 支持一系列性能标准的能力

语音压缩应用的变化造成了需求功能的大跨度。对于一些应用，例如移动通信，功率考虑和芯片面积可以作为设备的推动标准。对于其他应用，主要目的可以是一个高数据率系统。

5. 可扩展结构

要创建支持如此大范围设计标准的灵活性，需要开发一个可扩展结构，提高物理硬件的层次来匹配规格需要。推动可扩展结构的重点有：

1）需求数据率；

2）面积限制；

3）时钟速率限制；

4）功率限制。

图 10-18 所示为可扩展设计的可能结构。

6. 时钟速率性能

系统所需时钟速率来自于结构设计和目标技术。指定系统要求使设计师对目标技术作出选择，并便于在如功耗和面积的其他性能标准作出折衷。

7. 流水线级别

所需时钟速率可依赖于设计内的流水线以减少关键路径。流水线内的设计子模块的选择会对性能影响很大。例如，考虑一个既执行编码和又执行解码的功能模块，换句话说，它可以执行全双工操作。语音样本为 8kHz。多信道应用的所有信道的全部编码和解码的计算都需要在下一个样本前，也就是在 1/8000s (0.000125s) 内进行。对于最大时钟频率为 300MHz 的示例模块，在 0.000125s 时限内会有 37500 个时钟周期。在此示例中，如果认为全双工操作需要 20 个时钟周期，那么可能的通道数为$(300000000 \div 8000) \div 20 = 1875$ 个双向信道。那意味着 1875 个编码和 1875 个解码信道。

假设模块的通道更多，达到了 500Hz 时钟速率，每个双工操作的循环次数增加 25 次，则可估计双通道的数量为$(500000000 \div 8000) \div 25 = 2500$。

图 10-18 和图 10-19 所示为参数化的 ADPCM IP 核心需要支持的应用范围。

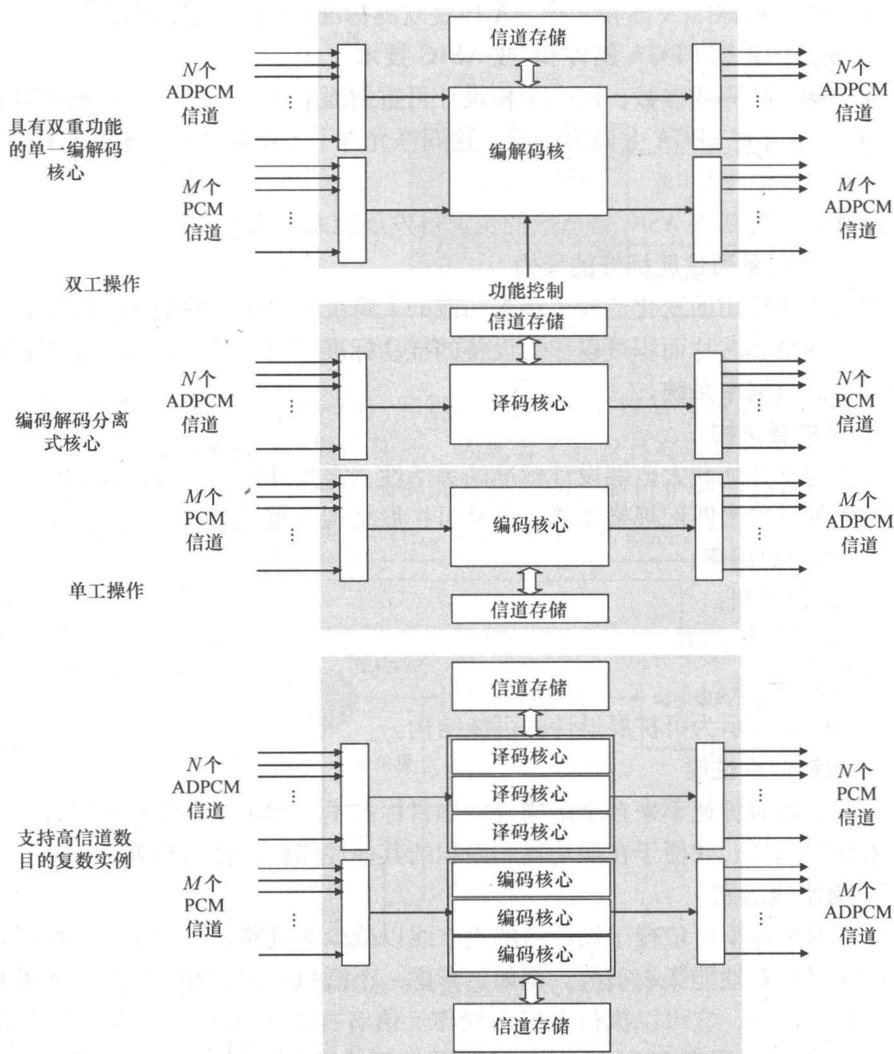


图 10-18 多信道 ADPCM

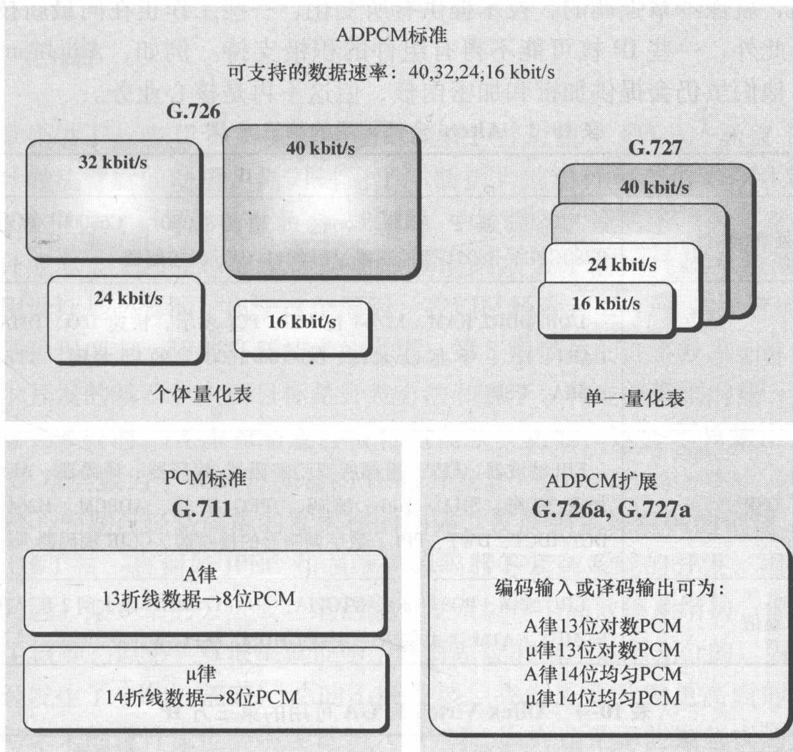


图 10-19　PCM 和 ADPCM 的区别

10.7　FPGA 的 IP 核

FPGA 公司早些时候就已确认 IP 核的重要性，协助制定使用他们的技术解决方案。出于这个原因，他们开始发展自己内部的 IP 库，但很快发现与第三方合作关系的好处，因为双方都能受益，一方面能从 IP 核中获得收入；另一方面是 IP 的可用性简化了客户解决方案。两大主要供应商都在运作 IP 程序，Xilinx 用 Alliance 程序，Altera 用 Megafunctions Partners 程序；Actel 有 CompanionCore 联盟计划，Lattice 有 ispLeverCORE™ 连接。在任何情况下，公司必须满足大量标准，成为各自 IP 程序的成员。

表 10-3 和表 10-4 给出了可用于处理器、接口、DSP 和通信的第三方 IP 的全部范围。从表中可以看出，软核可用于至少一个形式的微处理器或微控制器。大范围的接口技术是极其重要的，使 FPGA 能够实现电信方面的应用。DSP 列表包含了这本书中列出的大部分功能，以及如 JPEG 和 MPEG 的复杂系统。

当然，就像本章强调的，技术提供有所变化，一些工作正在向最新技术的核心迁移。此外，一些 IP 核可能不再有组件的积极支持。例如，Amphion 被科胜讯收购，他们虽仍会提供加密和加密内核，但这不再是核心业务。

表 10-3 Altera 全部可用的第三方 IP

技术领域	可使用 IP 核
处理器	NIOSII、基于 ARM Cortex 处理器、2901、C68000 微处理器、CZ80CPU、R8051 微控制器、DF6811 CPU 微控制器
接口	DDR/DDR2 RAM、32/64 位 PCI、PCI 表示、快速 I/O、DMA、CAN、UART、I2C、千兆以太网、10/100/1000/5 的以太网、ATA - 4/5、AMBA、USB
DSP	FIR 滤波器、FFT、里德所罗门解码器/编码器、译码器、AES 和 DES 加密/解密、SHA、Turbo 编码、JPEG 格式、ADPCM、H264、JPEG、DCT/IDCT、DWT、PPL、数字调制器的接收器、CCIR 编码器/解码器
通信	CRC 编译、POS - phy，UTOPIA，SONeT/SDH、以太网 2 层、千兆以太网 MUX、ATM 格式/变形、AA5、HDLC、SPI - 4

表 10-4 Xilinx Virtex FPGA 可用的第三方 IP

领域	功能	提供商
加密/解密	SHA、MD5 Hasing、AES AES 加密/解密、TDES、RSA	Helion Tech. Limited Hidea Sol. Co. , Ltd，CAST，Inc.
视频	H. 264/AVC 译码、分块 H. 264/AVC 分块 JPEG 编码	Nero AG，Global Dig. Tech. Elecard Dev. CJSC CAST Inc. , Barco. Silex
音频处理	样本速率转化为 32 位 APS3 处理器	Coreworks CAST，Inc.
计算机 电话通信	SDRAM 控制 Flash、SD 存储控制器 DDR 控制器 串行 ATA 8 位高速控制器 SONET/SDH 帧调节器 数字封装从设备 ADPCM 控制器 以太网 MAC HiGig Interlaken 互连	Array Electronics Eureka Technology HCL Tech. Ltd. , CAST Inc. ASICS World Ser. Ltd CAST Inc. Xelic，Inc. Beckhoff Auto. GmbH Pinpoint Sol. Inc. MorethanIP GmbH Sarance
数据压缩	LZRW3	Helion Tech. Ltd

10.8　总结

本章提供了一些 IP 核的关键问题的概述，并着重于 FPGA。涵盖了 IP 核和重用设计做法背后的动机，同时讨论了不断扩大的设计生产力差距。这一点在 ITRS 2005 报告（半导体行业协会　2005）中被强调，说明了“为了避免成倍增长的设计成本，大部分芯片设计的功能规模必须大于前一代技术的两倍”。为此，重用设计是必要的。出于这个原因，我们已经看到 IP 行业的显著增长。

在某种程度上，随着高新技术的发展，第 8 章描述的许多技术，已经成为这样的技术开发的基石。主要目标是开发方法和源于一种电路架构的流，当把算法描述传输到硬件时，规律和缩放等被自然捕获。然而，这并不只是出于这个原因，第 12 章会致力于 RLS 过滤器的 IP 核开发。

IP 核心问题已经从简单的算术核的创建，发展到系统组件的创建，如 FFT，DCT 和 DWT 核，直到如 JPEG 和 MPEG 编码器等复杂系统的开发，因此这带来了设计问题。第 8 章的方法成功地创建了复杂组件的高效体系结构，像 FIR 滤波器、DCT 核等，但第 9 章很快就证明了这对系统的描述是不充分的。因此，设计问题已经发生了变化，随着问题的不断发展，我们需要开发更高级的技术。然而，任何未来的设计流也必须尝试纳入 IP 核，因为这是不能被忽视的大量设计工作。因此，下一章目的是解决这个问题。

参 考 文 献

Alaraje N and DeGroat J (2005) Evolution of re-configurable architectures to SoFPGA. *48th Midwest Symp. on Circuits and Systems* pp. 818–821.

Association SI (2005) International technology roadmap for semiconductors: Design. Web publication downloadable from http://www.itrs.net/Links/2005ITRS/Design2005.pdf.

Association SI (2006) International technology roadmap for semiconductors: Design and systems drivers. Web publication downloadable from http://www.itrs.net/Links.

Birnbaum M (2001) VSIA Quality Metrics for IP and SoC. *Int. Symp. on Quality Electronic Design*, pp. 279–283.

Birnbaum M and Sachs H (1999) How VSIA answers the SOC dilemma. *Computer* 32(6), 42–50.

Chiang S and C. T (2001) Foundries and the dawn of an open IP era. *IEEE Computer* 34(4), 43–46.

Coussy P, Baganne A and Martin E (2002) A design methodology for IP integration. *Proc. Int. Symp. on Circuits and Systems*, pp. 127–130.

Coussy P, Casseau E, Bomel P, Baganne A and Martin E (2006) A formal method for hardware IP design and integration under I/O and timing constraints. *ACM Transactions on Embedded Comp. Syst.* 5(1), 29–53.

Coussy P, Casseau E, Bomel P, Baganne A and Martin E (2007) Constrained algorithmic IP design for system-on-chip. *Journal of Integr. VLSI* 40(2), 94–105.

Craven S and Athanas P (2007) Examining the viability of FPGA supercomputing. *EURASIP Journal on Embedded Systems* 1, 13–13.

Davis L (2006) Hardware Components, Semiconductor–Digital–Programmable Logic IP Cores. Web publication downloadable from http://www.interfacebus.com/IP_Core_Vendors.html.

Ercegovac MD and Lang T (1987) On-the-fly conversion of redundant into conventional representations. *IEEE Trans. Comput.* 36(7), 895–897.

Erdogan A, Hasan M and Arslan T (2003) Algorithmic low power FIR cores. *IEE Proc. Circuits, Devices and Systems* 150(3), 155–160.

Gajski D, Wu AH, Chaiyakul V, Mori S, Nukiyama T and Bricaud P (2000) Essential issues for IP reuse. *Proc. ASP Design Automation Conf.*, pp. 37–42.

Guo JJ, Ju R-C and Chen J-W (2004) An efficient 2-D DCT/IDCT core design using cyclic convolution and adder-based realization. *IEEE Trans. Circuits and Systems for Video Tech.* 14(4), 416–428.

Howes J (1998) IP New Year. *New Electronics* 31(1), 41–42.

Huang Y, Cheng W, Tsai C, Mukherjee N, Samman O, Zaidan Y and Reddy S (2001) Resource allocation and test scheduling for concurrent test of core-based SOC design. *Proc. IEEE Asian Test Symposium (ATS)*, pp. 265–270.

Hunter J (1999) *Rapid Design of DCT cores*. PhD Dissertation, School of Electrical and Electronic Engineering, Queen's University of Belfast.

Hwang K (1979) *Computer Arithmetic: Principles, Architecture and Design*. John Wiley & Sons, Inc., New York, NY, USA.

IRTS (1999) *International Technology Roadmap for Semiconductors*, 1999 edn. Semiconductor Industry Association. Web publication downloadable from http://public.itrs.net.

Junchao Z, Weiliang C and Shaojun W (2001) Parameterized IP core design. *Proc. 4th Int. Conf. on ASIC*, pp. 744–747.

Koren I (1993) *Computer arithmetic algorithms*. Prentice-Hall, Upper Saddle River, NJ, USA.

Lightbody G, Woods R and Francey J (2007) Soft IP core implementation of recursive least squares filter using only multiplicative and additive operators *Proc. Int. Conf. on Field Programmable Logic*, pp. 597–600.

McCanny J, Hu Y, Ding T, Trainor D and Ridge D (1996) Rapid design of DSP ASIC cores using hierarchical VHDL libraries. *30th Asilomar Conf. on Signals, Systems and Computers*, pp. 1344–1348.

Moretti G (2001) Your core, my design, our problem, *EDN*, 10 November 2001, pp. 57–64.

Rowen C (2002) Reducing SoC simulation and development time. *Computer* 35(12), 29–34.

Schwarz EM and Flynn MJ (1993) Parallel high-radix nonrestoring division. *IEEE Trans. Comput.* 42(10), 1234–1246.

Sekanina L (2003) Towards evolvable IP cores for FPGAs. *Proc. NASA/DoD Conf. on Evolvable Hardware*, pp. 145–154.

Shi C and Brodersen R (2003) An automated floating-point to fixed-point conversion methodology. *IEEE Int. Conf. on Acoustics, Speech, and Signal Proc.* 2(2), 529–32.

Soderquist P and Leeser M (2004) Technology Roadmap for Semiconductors. *IEEE Computer* 37(1), 47–56.

Srinivas H and Parhi K (1992) A fast VLSI adder architecture. *IEEE Journal of Solid-State Circuits* 27(5), 761–767.

Takagi N, Yasuura H and Yajima S (1985) High-speed VLSI multiplication algorithm with a redundant binary addition tree. *IEEE Trans. Comput.* 34(9), 789–796.

University N (2007) Variable precision floating point modules. Web publication downloadable from http://www.ece.neu.edu/groups/rpl/projects/floatingpoint/index.html.

Varma P and Bhatia S (1998) A structured test re-use methodology for core-based system chips. *Proc. IEEE Int. Conf. on Test.*, pp. 294–302.

VSIA 2007 Vsia quality IP metric. Web publication downloadable from http://www.vsia.org/documents/.

Walke R (1997) High sample rate givens rotations for recursive least squares. PhD Thesis, University of Warwick.

第11章　基于模型的异构FPGA设计

11.1　引言

针对这一点的材料明确指出了基于FPGA的DSP系统更高级别表示的转变要求和对此进行优化的需要。第6章的材料涵盖的较低级别的设计技术旨在生产高效的FPGA应用电路架构。这些设计技术涵盖的不只是选择如何把内存要求映射到LUT和嵌入式RAM资源，还有对性能的影响，也使用分布式算法和可重构MUX来降低硬件成本。然而，对于粗粒度异构平台的FPGA技术演变，即由处理器和涉及专用乘数和MAC单位的复杂DSP块组成，已经否定了很多后来技术的影响。

第8章的材料和第9章的后续工具开发工作，说明了如何探索DSP系统的并行和流水线水平的SFG和DFG描述，而不是较低级的基于HDL的电路结构。它表明在FPGA技术可行的处理和内存资源的考虑下，如何调整并行和流水线的水平至最好地匹配应用的性能需求。这是假设SFG表征有效表示了一个DSP系统计算的需求，但这不是个案。例如，一个系统应用需要不同位置的大量FIR滤波器；SFG的限制需要每个滤波器有单独的硬件，除非在SFG表征中已显示创建硬件共享。它不应该是如何描述系统的问题，而是用户可以调查的系统优化问题。

因此，显然需要比SFG更高层次的表征来允许用户调查系统级分区和系统级优化的影响，如只是被提出，仍有待探索的硬件共享。这种方法形成了基于计算MoC大趋势的一部分。MoC被用来表达系统特点，如时效性，即该系统如何处理时间、并发性、活性、异质性、接口和反应性（Lee和Sangiovanni-Vincentelli　1998）的概念。他们试图定义一个系统如何运转，并与真实和模拟世界互动的原理。为严格模拟不同类型的嵌入式系统提出了大量的模型。为特定类型的系统建模，应该基于该系统的具体特点来确定合适的MoC。例如，DSP系统的一般特征可以描述系统输入数据流的重复密集计算。因此，DSP系统开发人员经常在这一选择上得出相同的结论，并使用数据流MoC（Najjar等　1999）。本章将概述使用最广泛的数据流域及嵌入式DSP系统的系统级设计和优化的方式。

本章将给出可以用于创建基于FPGA的DSP系统的详细数据流MoC分析，并用创建此类系统功能和使用恰当例子进行演示来说明。11.2节为数据流建模

概述，将重点介绍一些不同的类型，特别涵盖了基于 FPGA 的 DSP 系统快速实现的挑战。11.3 节将阐述 DFG 描述的变化如何对嵌入式系统应用产生重大影响。在 11.4 节中，将给读者使用流水线核的系统描述的综合，且无需重新设计核。11.5 节将概述必要的控制和封装核的设计。归一化格型滤波器和固定波束形成器的两个例子将被用于 11.6 节的方法演示。最后是 11.7 节的总结。

11.2 数据流建模及快速实现基于 FPGA 的 DSP 系统

最受欢迎的通用数据流语言源于卡恩进程网络（Kahn Process Network，KPN）模型（Kahn 1974）。KPN 模型描述了通过单向 FIFO 队列进行的一套并行进程（或“计算站”）通信。计算站使用本地化内存从输入行读取数据符号，产生一个或所有输出行上的输出。在 DSP 系统中，符号通常是被数字化的输入数据值。对系统的连续输入生成输入数据流，促使计算站产生系统输出数据流。KPN 总体结构如图 11-1 所示。每个 KPN 输入样本的具体计算功能重复应用的语义，使建模与 DSP 系统性能达到良好匹配。

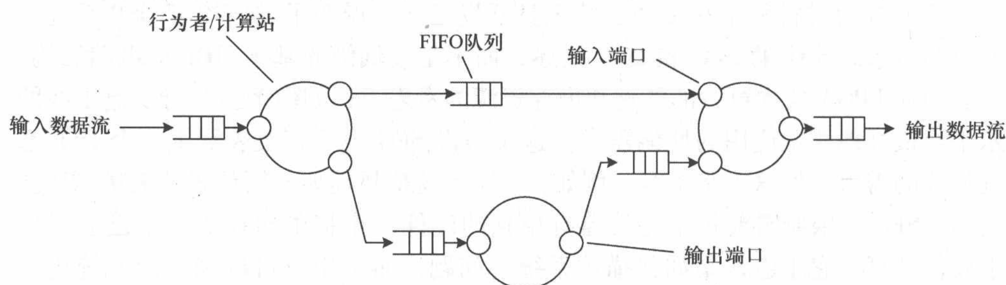


图 11-1 简单的 KPN 结构

在数据流进程网络（Dataflow Process Network，DPN）模型（Lee 和 Parks 1995）中，KPN 模型增强了计算站（在这里被称为触发器）性能的语义。触发器触发序列被定义为特殊类型的卡恩进程，称作数据流进程，每个触发器都表示输入标记到输出标记的映射，以及输入流到输出流的继承映射。一套触发规则为每个触发器规定了如何及何时进行触发。具体来说，触发器使用输入符号并产生输出符号。一套顺序触发规则为每一个触发器存在，定义了触发器可能会触发的输入数据条件。为了给出 DPN 描述触发器如何触发和确定性通信的坚实的计算基础，许多针对这个主题的应用程序的相关改进已被提出。同步数据流（SDF）、环静态数据流（CSDF）和多维同步数据流（MSDF）这三种特定变量是本节关键。现概述这三者的含义。

11.2.1　SDF

Lee 把同步数据流（Synchronous Dataflow，SDF）系统（Lee 和 Messerschmitt 1987b）定义为“我们可以预先指定调用块时，每次输入使用的样本数目和每次输出产生的样本数目”的系统。这不是提供一般 MoC 定义（Lee 和Sangiovanni - Vincentelli 1998）的同步性定义；然而，它允许静态系统调度的派生（即在编译时生成），其重要性在于这意味着可产生运行时耗低的多处理器调度的派生。这是数据流 MoC 的一项重大进展，为数据流系统建模与技术实现的研究主体开辟了道路。然而，由于 SDF 禁止依赖于数据的数据流行为，因此获取这种优势就要牺牲表现力。

SDF 图中每一个触发器 i 的端口 j 都有一个关联阈值 tf_{i_j} 来指定触发器触发端口使用/产生（根据端口方向）的符号数目。此值被相邻的端口引述，如图 11-2 所示。当 SDF 中所有端口阈值统一时，此图被认为是同质的，或是单速率 DFG（Single Rate DFG，SRDFG）。否则，DFG 称为多速率 DFG（Multi - rate DFG，MRDFG）。

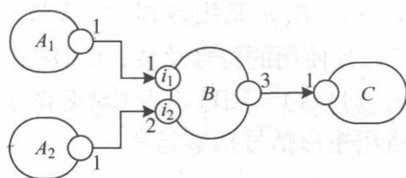


图 11-2　简单的 SDF 流程图

如果假设触发器 j 连接到 arc$i$，在触发器第 n 次调用中，$x_j^i(n)$是产生的符号数目，$y_j^i(n)$是使用的符号数目，则 SDF 模型可以用拓扑矩阵 $\boldsymbol{\Gamma}$ 描述（见式(11-1)）。

$$\boldsymbol{\Gamma}_{ij}=\begin{cases}X_j^i(n) & \text{如果触发器 } j \text{ 产生 arc} i\\ -Y_j^i(n) & \text{如果触发器 } j \text{ 消耗 arc} i\\ 0 & \text{其他}\end{cases} \tag{11-1}$$

证明 DFG 一致性的一个关键因素是满足一系列平衡方程（Lee 1991）。触发器 A 在调度迭代中触发比例为 $\boldsymbol{q}_{(\mathrm{A})}$，并在每一次触发中产生符号 t_{A}，触发器 B 触发比例为 $\boldsymbol{q}_{(\mathrm{B})}$，并在每一次触发中使用 t_{B}。如果 A 连接到 B，则在调度迭代中 FIFO 队列返回到初始状态（Lee 和 Messerschmitt 1987a），式（11-2）将成立。把图中的每一个 arc 的方程集中起来，构建一个平衡方程系，并简写为式(11-3)。在式（11-3）中，重复矢量 q_i 描述了当执行图中调度的迭代时，每一个触发器的触发数目 i。

$$\boldsymbol{q}(A)t_{\mathrm{B}}=\boldsymbol{q}(B)t_{\mathrm{B}} \tag{11-2}$$

$$\boldsymbol{\Gamma}q=0 \tag{11-3}$$

11.2.2　CSDF

环静态数据流（Cyclo－Static Dataflow，CSDF）模型（Bilsen 等 1996）指出 MRDFG 的局限性在于，其仅支持触发之间恒定的触发器触发。它扩展了 SDF 模型，允许触发器触发的周期性变化，同时仍然保持静态调度功能。每个触发器的触发活动产生由每个触发器 v_j的 j 触发，序列 $\gamma = [f_j(1), f_j(2), \cdots, f_j(P_j)]$，都有不同的触发规则。$\gamma_i$ 是触发器的第 $i(\bmod\ j)$ 个触发，产生反复循环的触发序列。SDF 图中的产生和使用标量值都被替换为长度 P 的向量，且定义 P_i 为触发器第 i 个端口生产/使用的符号数目。如果假设触发器 j 连接到 arc$i$，第一次 n 调用过程中，$X_j^i(n)$ 是产生的符号总数，$Y_j^i(n)$ 是使用的符号总数，CSDF 拓扑矩阵 $\boldsymbol{\Gamma}$ 的方程定义如式（11-4）。图11-3为 CSDF 图示例，相邻端口的阈值向量用矩形括号括了起来。

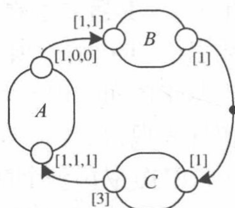


图 11-3　简单 CSDF 流程图

$$\boldsymbol{\Gamma}_{ij} = \begin{cases} X_j^i(P_j) & \text{如果触发器 } j \text{ 产生 arc}i \\ -Y_j^i(P_j) & \text{如果触发器 } j \text{ 消耗 arc}i \\ 0 & \text{其他} \end{cases} \tag{11-4}$$

11.2.3　MSDF

SDF 原子标记原则意味着多维系统（具有多维符号计算，如向量或矩阵）必须使用一维流在图上铺开，此限制可以隐藏数据级并行算法。为了克服这种限制，SDF 模型被推广到多维同步数据流（Multidimensional Synchronous Dataflow，MS-DF）。初期工作把 SDF 域（Lee 1993a，b）演变成矩形点阵采样问题，之后又将技术扩展为任意形状点阵（Murthy 和 Lee 2002）。标记的生产和使用现在被指定为 M 元，每一个 arc 的平衡方程数量从 1 增加到 M。产生和使用 M 维符号的示例 MSDF 流程图及其关联平衡方程如图 11-4 所示（Lee 1993a），圆括号中给出相邻端口。此域为 SDF 图中的多维调度问题提供了巧妙的解决方案，并公开了高维度符号的并行附加内部符号（Lee 1993a）。

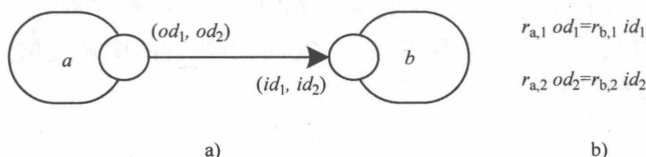


图 11-4　MSDF 流程图与相关平衡方程
a）流程图　b）平衡方程

11.2.4　数据流异构系统原型

通常情况下，为 SoC 和 FPGA 设计异构系统需要从高层系统模型到应用增量精化。这就需要解决通信改进、专用硬件合成及其他大量问题。或者通过将应用的行为语义密切匹配到高级规范语义，即启用从功能性到执行域的直接翻译。这是一种快速的系统集成技术。图 11-5 所示为所需方法论的一般概述，并展示了该方法的三个关键要求：

1）合适的规范建模域；
2）快速的集成方法；
3）分析能力和高级别优化执行能力。

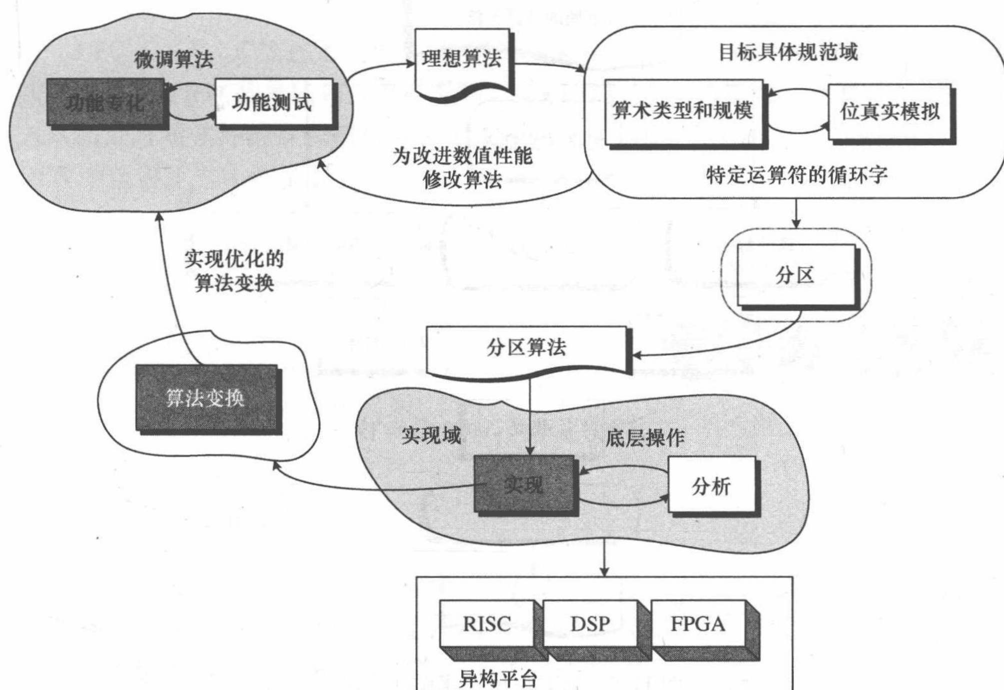


图 11-5　DSP 系统设计方法

高级算法模型在理想情况下要使用无限精确的运算，但在现代工作站技术的体系结构中，实现建模意味着在如 MATLAB 的建模应用中，通常使用整型及单/双精度浮点型运算就足够了。理想算法验证之后，数值建模是将适用于低精度的数值模拟（固定/浮 - 点）和字长最小化，以便为专用硬件组件的实现确定候选函数。这种类低精度数学建模操作的重要性被承认，并被如 C 语言的系统设计语言（Grötker 等　2002）和如 MATLAB 的工具所吸纳。

系统功能接着被映射到一个嵌入式平台上的物理处理器。此分区通常基于物理的实施限制因素，如吞吐量要求、可行专用硬件资源、通信带宽或电力消费。分区之后，开始实施。

11.2.5 分区算法实现

执行需要从分区算法到硬件、软件部分相互作用网络的快速转换。这种集成方法的总体结构如图 11-6 所示。分区算法在一个或多个微处理器、FPGA 和处理器间通信点上指定实现的功能池。这从功能级描述了整个嵌入式系统。此描述需要被转换为物理表述。触发器在解决三个主要问题之前被收集到池中：

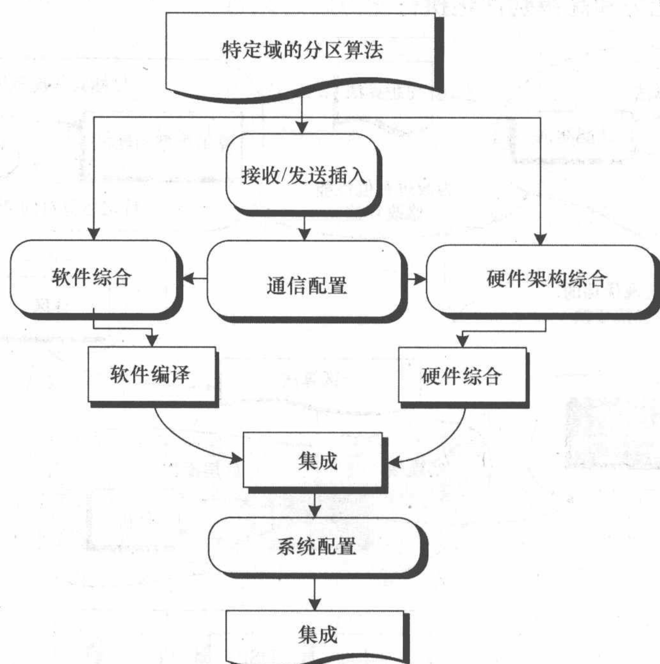


图 11-6　DSP 系统快速设计实现

1）嵌入式微处理器的软件集成；

2）每个 FPGA 的硬件合成；

3）处理器间嵌入的通信结构。

为使设计时间最小化，设计者可以使用广泛的、不断增长中的复合功能库，以有效实现硬件和软件的特定触发器。特别是 FPGA 设计者，目前存在大量的 IP 组成部分（芯）库，如字大小的参数化功能，鼓励重复利用多个系统实例。由于使用这些组件可减少设计时间，因此这种方法强调核心基础的设计，因为此应

用减少了核心集成。

然而，这种做法有大量的的问题。将来自多个不同源的多个不同内核集成到一个单一的 FPGA 核心网络，意味着可能会出现接口问题。尽管如 VSIA（VSIA 2007）的标准机构作出了努力，但仍没有被广泛采用的核心标准化接口协议，这意味着设计师必须将特定的应用加上去。

此外，该研究方法使用的基于 MoC 的设计法，表现的灵活性可能是核的需求，而不是核集成过程。本章后面会有更多详细信息。本节其余部分将介绍处理器间通信行为改进和软件合成的方法。

11.3　DFG 嵌入式软件的快速合成与优化

DFG $G=V,\ E$ 描述了由一组边 E 相互关联的一组顶点（触发器）V。边从概念上讲是 FIFO 符号队列，其符号可以是任意数值存储单元。从算法转换到嵌入式微处理器软件的常规执行有 4 个主要阶段，如图 11-7 所示。图 11-8 所示的 DFG 中包含了这 4 个步骤。

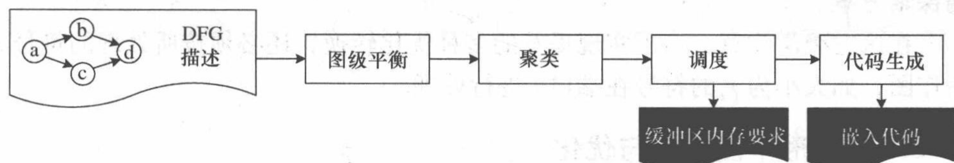


图 11-7　DFG 的快速软件综合

11.3.1　图形级优化

第一级的优化是将算法工程技术应用于 DFG 执行优化。这是特别适用于使用 DFG 语义的系统建模（Murthy 和 Lee 2002）。图11-9 所示为简单等效 DFG。假设符号有原子性，

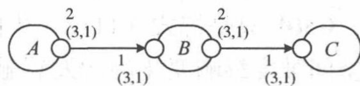


图 11-8　DFG 示例

则图 11-9a 的最大可能分区图是单个处理器，并转换所有通信频道实施矢量。这隐藏了一些向量乘法算法的固有数据并行性能。对于一个 n 元向量，处理向量乘法可能需要 n 个周期，这成了实现高吞吐量的重大缺陷。或者可以为标量指定符号如图 11-9b 所示，使多处理器的分区和相应的吞吐量增加 n 倍。这种操作也改变了每个处理器所需的内存资源，因为它们现在都需要标量符号缓冲，而不是 n 元向量。缺点包括使用内部符号的并行性时，调度复杂性增加。由于 FPGA 计算资源的结构是在设计者的控制之下，这一优化可能特别适用于 FPGA 硬件组件网

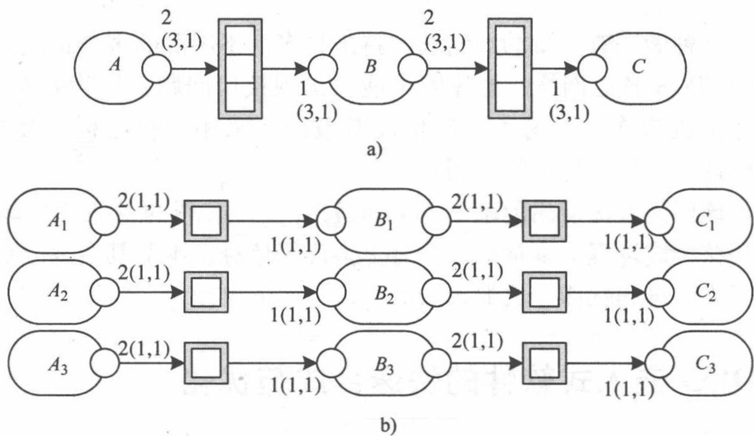


图 11-9 标量/矢量 DFG 比较

a）DFG 的 3 个矢量符号元素 b）DFG 标量符号

络合成。因此，这可能会为 DFG 触发器网络映射到 FPGA 提供非常有前景的结构探索方案。

在这一级需注意，为了实现提及的多种优化转换，还必须按所处理的符号灵活作图，如大小为 x_i 的符号在端口 i 进行处理。

11.3.2 图形平衡操作与优化

在基于数据流的快速嵌入式系统设计领域中，SDF 域的建立与最少嵌入式执行时间的相关潜力，无疑是最重大的两个进展，还有 DFG 确保内存约束和无锁死执行（Lee 1991）的一致性的形式化分析方法的介绍。本节中所有其他更具表现力的数据流域都源于这些想法的泛化，特别是式（11-2）和式（11-3）。例如，CSDF 只是把式（11-2）中的标量 q 和 r 概括为二维对象，并利用额外提供的空间维度影响变化的触发器触发活动，在不违反式（11-2）和式（11-3）一致性条件的情况下，利用之前隐藏的并行性，使内存资源得到更有效的利用。

第一阶段中，从 DFG 应用程序到嵌入式执行的转化，执行图形平衡，即满足式（11-2）和式（11-3）（Parks 等 1995）的一致性条件。例如图 11-8 所示的 SDF 图，有着图 11-10 所示的触发器与编号，式（11-5）和式（11-6）分别给出的 $\boldsymbol{\Gamma}$ 和 $\boldsymbol{q}$。

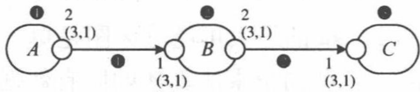


图 11-10 分值范例 DFG

$$\boldsymbol{\Gamma}=\begin{bmatrix}2 & -1 & 0\\ 0 & 2 & -1\end{bmatrix} \tag{11-5}$$

$$q^{T} = [1 \quad 2 \quad 4] \tag{11-6}$$

在抽象的平衡层，由于应用模块（Lee 和 Messerschmitt 1987a）的处理，主体优化机会增加。从根本上说，这包括应用程序的整数比因子 q 以整数缩放了每个触发器的触发数目。应用较低级转换，可以提高执行效率和性能，如第 11.3.4 节所述。

11.3.3 聚类操作和优化

图 11-8 所示的 DFG 示例存在众多的有效调度，$S_1 = ABCCBCC$ 和 $S_2 = ABBCCCC$是两个有效的例子。需注意，这两个调度的触发器触发有不同序的分组。在给定的数据流算法的均衡表示中，q_i 定义了迭代调度中触发器的触发数目。在聚类的阶段，每个触发器的触发被细分成成组的一个或多个连续触发，称为执行。每一个触发器 i 执行的触发数目被称为它的粒度，它定义了执行之前的触发器端口的可输入标记数目。如图 11-11 所示的调度 S_1 和 S_2，给出了概述每一个触发器触发和执行的依赖图。

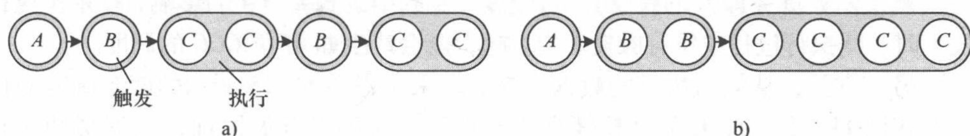


图 11-11 调度触发/执行组织

a）S_1 独立流程图 b）S_2 独立流程图

从执行阈值和调用方面诠释式（11-2），可定义如式（11-9）所示的平衡系统，作为一个平衡系统的定义，$\boldsymbol{\Gamma}_e$ 在式（11-7）中给出，通过求解每一个 DFG 的式（11-8），推导出执行重复向量 $\boldsymbol{q}_e$。需特别注意缩放因子 k_j 与每个触发器 j 相关联。在本节稍后会将其用于聚类阶段的优化。

$$\boldsymbol{\Gamma}_{ij} = \begin{cases} k_j \times x_j^i\ (n) & \text{如果触发器 } j \text{ 产生 arc} i \\ -k_j \times y_j^i\ (n) & \text{如果触发器 } j \text{ 消耗 arc} i \\ 0 & \text{其他} \end{cases} \tag{11-7}$$

$$\boldsymbol{q}_e(j) = q(j)/k_j \tag{11-8}$$

$$\boldsymbol{\Gamma}_e q_e = 0 \tag{11-9}$$

满足式（11-9）的调度采样率一致，是有效调度的两个关键特征（Lee 1991，详见第 11.3.4 节）之一。因此通过操作 $\boldsymbol{\Gamma}_e$和 $\boldsymbol{q}_e$来控制的聚类阶段，提供了执行优化的两个自由度。此操作受到与每个触发器在聚类阶段相关联的操纵因子 k 的影响。需考虑如何实现图 11-8 所示的 DFG 的执行优化。图 11-10 中标记

的 $\boldsymbol{\Gamma}_e$和 $\boldsymbol{q}_e$触发器的值在表 11-1 给出。

表 11-1　示例调度 Γ_e和 q_e

调度	$\boldsymbol{\Gamma}$	$\boldsymbol{q}_e^T$
ABCCBCC	$\begin{bmatrix} 2 & -1 & 0 \\ 0 & 2 & -2 \end{bmatrix}$	[1　2　2]
ABBCCCC	$\begin{bmatrix} 2 & -2 & 0 \\ -2 & 4 & -4 \end{bmatrix}$	[1　1　1]

高 g 变换通过增加 k_j在式（11-8）和式（11-7）中的值来最大化单个触发器执行阈值。这减少了每个触发器的执行数量，用可以减少应用执行的运行时耗的方法集中触发器。这些间接消耗可能有各种来源，如 IPC（远程处理器设置/关闭通信链路时可能出现处罚时）或动态调度。考虑图 11-11a 所示的第一调度 $A(2B(2C))$，将数据发送到远程处理器，其中触发器 C 为 IPC 触发器。在这种情况下，设置和关闭转换的相关系统开销分别被触发器执行的调用和结论调用。若要最大化通信效率（即每次损失调用的传输数据量），则应通过增大 k_c的值最大化每次执行时 C 触发的数量（即 C 的粒度）。这受聚类步骤中设置 $k_c=4$ 的影响，C 单次执行发生时，此执行过程中所有触发发生。这可能会产生如 *ABBCCCC* 的调度。

另一方面，从初始执行的数据内存需求来考虑图 11-15 所示的两个调度的执行。在图 11-11a 中，C 的高粒度意味着在第一次触发开始之前，数据缓冲区必须保存将被输入到 C 的所有 4 个符号。在这种情况下，为了减小执行所需的缓冲内存，可使用高 q 变换。在图 11-15 的情况下，这涉及最小化 k_c 和 k_b，如 $k_c=2$，$k_b=1$。这减少了执行中的触发数目，也减少了 b 和 c 开始执行前必须存储的符号数目。

11.3.4　调度操作和优化

生成 DFG 的嵌入式应用源代码的最后一步为执行的调度。在此步骤中，每个触发器的各种执行按执行顺序排序。如其所述，确保 DFG 嵌入式执行可实现定期受理调度（Lee 1991），必须满足两个关键步骤。第一步，图形平衡期间核查图形样本率的一致性，如第 11.3.2 节所述。第二步是确保避免锁死现象。

对于嵌入式应用优化的不同方面的各种调度技术的研究，如数据/代码内（Bhattacharyya 等 1999）、IPC 或处理器间同步（Sriram 和 Bhattacharyya 2000），已投入了大量的研究工作。例如，如果没有在系统上执行交错调度优化的能力，则第 11.3.3 节所述的高 q 值转换技术是多余的。图 11-11a 所示的高 q 值的调度，分割 B 和 C 的执行并不影响数据内存需求，除非调度步骤可以识别进行交替 B 和 C 执行的可能性，这样用来存储 C 的输入符号的数据缓冲区就可以被执行重用，从而减少 C 一半的缓存需求。没有此交错调度，高 q 值优化并不影响嵌

入式软件的数据缓冲区要求。

11.3.5　代码的生成操作和优化

图 11-8 所示的 DFG 例子中给出 q，其有效调度包括 $S_1 = A(2B(2C))$；$S_2 = ABCBCCC$；$S_3 = A(2B)(4C)$；$S_4 = A(2BC)(2C)$。S_1，S_3 和 S_4 圆括号中的条件表明这些调度的循环调用。因此 S_1 有一个 A 触发，然后是两个重复的 BCC 触发。同样地，S_3 实现 $ABBCCCC$，S_4 实现 $ABCBCCC$。每个触发器实例处，代码生成器插入代表触发器功能的代码部分。例如，在 S_2 中，代码生成器必须把 7 个代码段分别实例化。然而，S_1，S_3，S_4 括号中的条件被换为代码循环的单一实例化，减少了调度触发器实例的数量。图 11-12 中的代码段是可从 S_4 生成的典型例子。不同调度方法生成的调度需要不同的代码内存空间。同样，每个调度的缓冲区记忆也不同。表 11-2 总结了每个调度的代码和缓冲区内存需求。很明显，系统如何调度对嵌入式表现有重大的影响。

```
code block for A
for (i=0;i<2;i++){
        code block for B
        code block for C
        }
for(i=0;i<2;i++){
        code block for C
        }
```

图 11-12　调度 S_4 生成的循环调度源码

表 11-2　调度不同的存储需求

调度	代码存储容量（模块）	缓冲区内存要求
$A(2B(2C))$	3	4
$ABCBCCC$	7	5
$A(2B)(4C)$	3	6
$A(2BC)(2C)$	4	5

11.3.6　系统级别设计方案探索中 DFG 触发器的可配置性

整合后，实时和物理约束的规范必须得到满足，如吞吐量、资源处理、内存使用情况或一些其他制约因素。只有满足这些约束，执行进程才是完整的。否则必须应用转换实现的方法。这些转换可以在低抽象级别中应用，其高设计细节意味着应用许多不同转换是困难而费时的，这在某种程度上否定了快速实施方法论的快速设计能力。另一种方法是应用对算法的高级别转换来影响嵌入式应用。这是一种粗粒度的探索，但明显比低级别转换更快，细节度更低，并可辅之以较低级别转换，来允许对应用的必要细粒度控制。这种快速设计探索是目前基于 MoC 快速实现方法的一个主要好处。

本节展示了数据流触发器在某些方面的灵活性如何使不同的多处理器调度，影响嵌入式的实现。这允许设计者在图形、平衡、聚类级别的嵌入式应用的系统级探索中操作 DFG。表 11-3 总结了各种类型的不同抽象级别优化所需的灵活性。

表 11-3 嵌入式优化 DFG 的参数配置

抽象层次	主要操作	描述
算法	A	多种参数
	X	连接弧度大小
	S	弧的参数
平衡	J	阻断因子
聚类	k	比例因子

第 11.3.1 节所述的图级转换取决于权衡带输入和输出弧符号维度的触发器数目的能力，它们在表 11-3 中分别被定义为符号 A 和 X。在平衡阶段，每个触发器的触发器数目可以通过执行模块处理进行操纵，例如按常数因子 J 缩放式（11-3）中的 q 矢量。在聚类阶段，触发器使用缩放因子来实现高 g/高 q 转换，如第 11.3.3 节所述。

11.3.7 DFG 专用硬件的快速合成和优化

第 6 章和第 8 章描述了 FPGA 上的寄存器可编程逻辑如何使它们适合于托管满足 DSP 功能的高吞吐量、流水线的电路架构。此外，大量针对把信号流图（SFG—SRDFG 的一类）转换到流水线硬件结构的自动化设计技术的研究，意味着这些设计方法可以实现高实时性能，并适用于以数据流为中心的系统级设计方法。这些类型的做法均符合快速应用方法论的快速核生成技术的需求。图 11-13 所示为把 MRDFG 规格转换为流水线电路应用的典型结构合成技术。

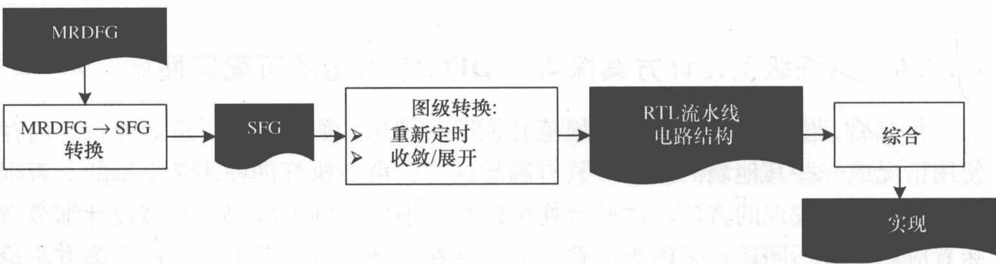


图 11-13 MRDFG 流水线核综合步骤

这个过程中最值得注意的是，要把算法的 MRDFG 描述转换为单一频率的 SFG 描述。这种转换是必需的，因为多速率模型（Lee 和 Messerschmitt 1987b）的行为语义明显比单速率更具表现力，且这些须被改进为使流水线硬件的低级别设计（Parhi 1999）可行。特别地，有大量与系统 SFG 建模相关的限制：

1）DFG 中所有端口的阈值整体固定；

2）每个触发器在调度迭代中只触发一次；

3）端口符号是原子性的。

SFG 的限制行为语义会比 MRDFG 产生更复杂的 MRDFG 模型，这些方面的相应操作表现为 SFG 图中的结构变化。结构本身的合成包括三个主要系统功能语法结构转换，产生一个适用于流水线的组件实现（Parhi1999）。时序重构过程早先在第 8 章介绍过，以及折叠和展开，用来优化分区或吞吐量约束的电路结构。

SFG 结构集成技术包括有先进调度算法的分层合成工艺和最低级（原始级）组件预定义内核的合成（Yi 和 Woods 2006）。然而，最近的工作，在这一领域中的大部分，在进行从系统功能语法翻译到流水线的硬件体系结构，很大程度上把 MRDFG 到 SFG 的转换看作是必要的前期步骤。图 11-13 所示的整个架构合成过程基于 SFG 域，生成的流水线核心体系结构是针对 SFG 结构的，因此通过逻辑扩展 MRDFG 的行为配置生成 SFG。这种情况下所施加的限制在第 11.3.8 节有表述。

11.3.8　专用硬件体系结构流水线的限制行为合成

将 SFG 合成方法应用于触发器的问题在于，表 11-3 中的任何配置因素的变更不会影响 MRDFG 的结构，却隐藏了语言的行为语义变化。然而，从 MRDF 到 SFG 转换的变化，会作为 SFG 中的结构性变化显现出来，因为此处的行为语义是固定的。流水线的核心是将结构合成技术应用于 SFG 的结果，这意味着，探索使用 MRDFG 的触发器配置变更的系统设计空间，需要为表 11-3 的因素所映射的设计空间中每个探索点重新合成流水线核心。为了说明产生这种限制的问题，图 11-14 所示为假设结构合成过程的 MRDFG。

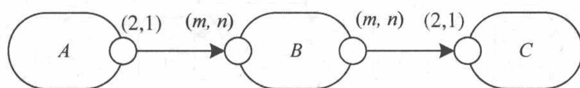


图 11-14　MRDFG 结构综合示例

在这里，触发器 A 产生弧上维度（2,1）的符号。这些符号被作为流水线的硬件组件 B 的消耗。B 消耗并产生维度（m,n）的符号，m 和 n 的值在设计者的算法级优选控制下，如第 11.3.1 节所述。MSDF 多维语义允许通过操纵语义符号维度探索，如第 11.2.3 节所述。然而当转换为 SFG 时，这种类型的并行内部符号的图形水平探索被禁止，并体现为 DFG 的结构变化。为了证明这一点，图 11-14 所示 SFG 的 B 的不同（m，n）值如图 11-15 所示。假定 SFG 触发器端口转移的符号维度是（1，1），G 值为 1，以确保 SFG 结构的变化只是由于符号维

度的变化。

在三种不同配置中，改变 DFG 符号维度就改变了 B 上的输入端口数目，因为每个 SFG 触发器必须把维度（1,1）的（m,n）转换为一个触发。此外，很显然，各种符号的维度受 q_i改变的影响，在图形调度表迭代中的触发器 i 的触发数目，导致触发器实例的数目变量，因为在 SFG 中，$q_{(i)}$始终是 1。因此，应用第 11.3.1 节所述的图级转换类型，而无需为开发空间中的每个点重新合成流水线硬件，触发器在端口数和触发实例数量方面必须是灵活的。第 11.3.2 节强调的基于图级平衡阶段模块处理来对 SFG 不同结构进行的分析，加强了这种观察。图 11-16 所示为图 11-14MRDFG 的 SFG 结果，$j=1$，$J=2$ 且符号维度为常数。这表示应用模块处理有效地复制了 SFG。

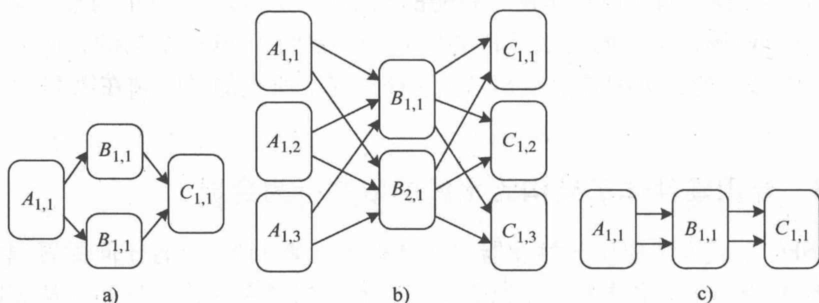


图 11-15 不同图层操作的 SFG 结构

a)（m,n）=（1,1） b)（m,n）=（1,3） c)（m,n）=（2,1）

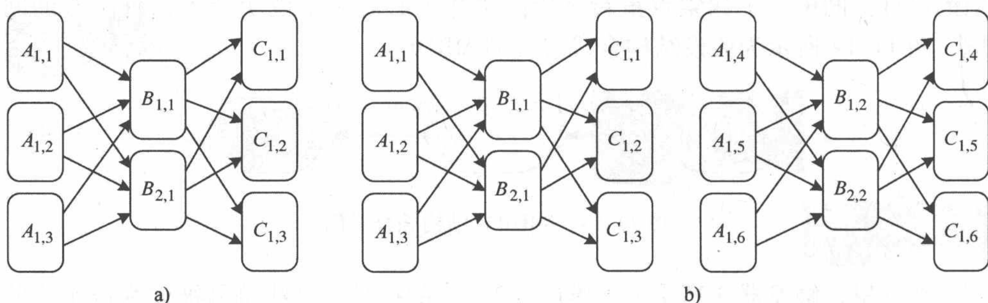


图 11-16 不同模块处理因素的 SFG 结构

a) $J=1$ b) $J=2$

这一节表明，由于 MRDFG 与 SFG 行为语义之间的差异，因此 SFG 不能用于多个 MRDFG 触发器配置。从逻辑上说，任何通过结构合成生成的 SFG 流水线核，不能用于不同 MRDFG 触发器配置，即不能重复用于多个 MRDFG 系统；

MRDFG 触发器的高级嵌入式系统开发中的每一次配置改变，都需要重新设计，重新合成一个新的流水线核心。

这并不只是 SFG 结构合成技术的缺点，而是所有嵌入式系统中，流水线核心的 MRDFG 生成的衍生结果。因此，当前的技术不能同时实现 MRDFG 框架下快速流水线的硬件组件合成，与基于重用生成组件的双重要求。当设计 MRDFG 级算法时，设计者只能推断，而不是直接施加控制专用硬件体系结构。下一节将概述克服了这些困难的流水线硬件核心的合成方法，使得生成的流水线专用硬件足够灵活，多个 MRDFG 触发器配置无需重新设计，使异构 FPGA 的 DSP 系统实现真正的系统级开发。

11.4　异构嵌入式 DSP 系统的系统级建模

设计者缺乏对应用结构的明确控制的关键问题在于，MRDFG 自身缺乏灵活性。标准数据流语言里的某个触发器，如 SDF 或 MSDF，可以表示任意数量的应用任务，而不是利用 DFG 触发器数目和解决方案中的核心数目之间的紧密关系。为了克服此结构的灵活性问题，可使用多维阵列的数据流（Multidimen Sional Arrayed Dataflow，MADF）域。为了演示域（McAllister 等 2006）的语义，考虑一个矩阵乘法问题——m_1 乘 m_2（维度分别是 (m,n) 和 (n,p)）。图 11-17a 所示为这一问题的 MSDF 图。

m_2 解释为一系列的平行列向量，其中每个向量是 m_2 的一列，向量可以分为任意大小的矩阵，允许 m_1 和数组 y 并行乘法，矩阵 m_{2_0}，m_{2_1}，…，$m_{2_{y-1}}$，m_{2_i} 分为 m_2 的 p 列向量 $i \times p/y$，…，$((i+1) \times p/y)-1$。图 11-17b 中，m_2 细分成 $p=4$ 的平行子矩阵。请注意常规乘法器数目和使用的每一个子矩阵大小之间的关系。这种关系可以被用来定期更改 DFG 结构，折衷触发器数目与每个接口处理的符号维度（即 MSDF 中同样的转变）同时仍然控制算法“结构”。给出一种适当的合成方法，能扩展以保证 FPGA 核心数目和符号维度，实例化核心数目变量的吞吐量用于交换资源。

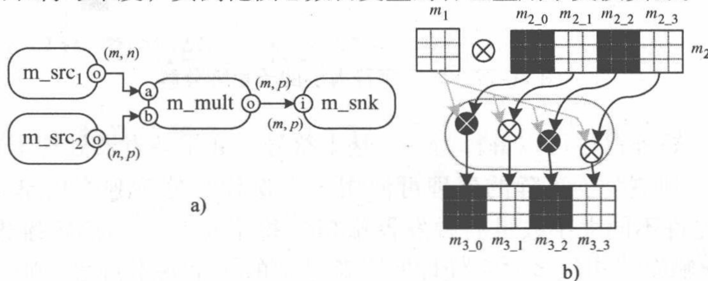


图 11-17　矩阵乘法 MSDF 和并行探究
a）MSDF 矩阵乘法器　b）并行矩阵乘法

MADF 模拟域被设计用于图形水平控制的底层处理结构。在 MADF 中，DFG 触发器和边缘的概念被推广到数组。MASDF 图中 $G = V_a$，E_a 描述了圆弧连接数组的触发器阵列。矩阵乘法的 MADF 图如图 11-18 所示。触发器数组是黑色，单一的触发器（或触发器数组大小为 1）是白色的。弧数组是实线，单一弧（或弧数组大小为 1）是虚线。触发器数组的大小是在触发器数组上的尖括号引用。

图中，系统设计者控制如图 11-18 所示的参数 y。它用来定义 m _ mult，m _ src$_1$，m _ src$_2$ 和 snk 触发器数组的大小，以及 m _ src$_2$ 上的端口 o，m _ mult 上的端口 b 和 o 所使用的符号维度。如果 m _ mult 触发器组转换成可匹配端口符号维度的一族核，那么这允许每个简单图级控制的核和符号维度数目。然而，如第 11.3.8 节所述，通过单率数据流合成法来制备的核，具有固定端符号维度。下面对如何克服这种限制进行概述。

11.4.1 交叉和模块触发器在 MADF 中的进程

图 11-18 所示为矩阵乘法实例中，m_2 的子矩阵数组输入的情况。单芯如何能做到足够灵活，以实现任何大小的输入矩阵从固定符号维度的 SFG 描述中生产流水线核？

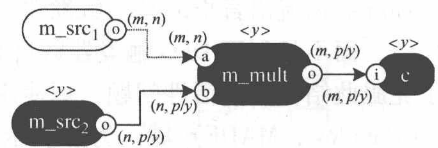


图 11-18 矩阵乘法器

如第 11.4 节中所述，每个子矩阵 y 可以解释一系列的 p 列向量，并由 m_2 的 $i \times p/y, \cdots, ((i+1) \times p/y) - 1$ 组成第 i 个子矩阵。以第 11.4 节为例，$y = 4$，第 i 个子矩阵可从图 11-19 所示的两个方面解释。

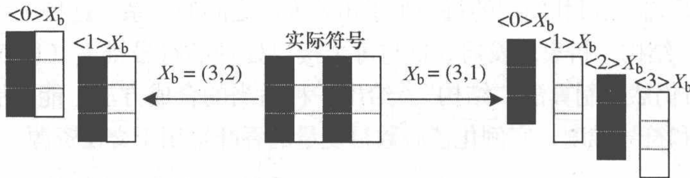


图 11-19 固定字符大小的子矩阵分解

这表明，矩阵符号可以解释为一族基本符号。如果触发器处理子矩阵只能处理基本符号，则整个子矩阵的处理可使用多个迭代来处理独立的基本符号。为此，MADF 允许不同大小数组的触发器端口，每个符号使用固定维度（基本符号）。若要使触发器迭代多次来处理实际符号中的多个基本符号，则 MADF 触发器可能是循环的（见第 11.2.2 节），单个触发使用通过数组中每个端口返回的一个或多个基本符号。图 11-20 所示为 MADF 矩阵乘法问题的完整固定符号处理

版本。请注意端口阵列（黑色）与固定符号维度的存在。

已经开发出内部符号的开发空间通过跨多个流运输基本符号分隔实际符号，并进一步执行开发。在端口阵列用于处理单个符号的情况下，可以实现每个端口交叉处理的数组，即使用单个基本符号，也能形成完整符号。在这种情况下，端口的每个数组元素的阈值是 1。然而，允许端口数组元素阈值大于 1，可启动模块处理，符号处理产生多流处理问题。在一个端口数组，第 i 个元素具有第 i 个以外的所有端口数组的大小为零的生产/使用的矢量。这些量表现为对角线的关系，即对于端口数组 a，除元素 0 外 a_0 的所有使用矢量都是零，除元素 1 外 a_1 的所有矢量都是 0 等。这种模式从广义上说，一个具有 n 个元素与阈值 z 端口的数组表示为 $<n>[z]$，如图 11-21 的 m_mult 所示，$y=3$。Z 的值用作每个子端口上的阈值，指示是否使用交叉或模块处理（$z=1$ 为交叉，$z>1$ 模块处理）。

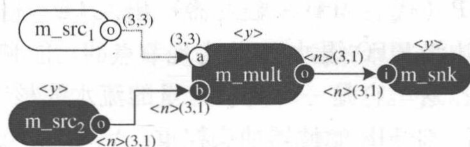


图 11-20　全 MADF 矩阵乘法

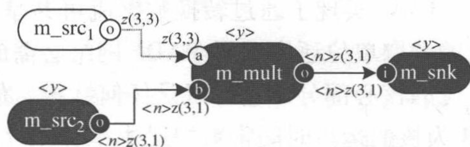


图 11-21　矩阵乘法的模块处理

鉴于 MADF 结构中这种水平的灵活性，很明显设计者可以通过合适的专用硬件合成方法来控制硬件组件的数量和每个图级水平的特点。生成 MADF 的嵌入式应用最紧迫的问题时是核心的结构合成方法可以连接流水线固定语义（即固定符号维度、阈值和处理流）之间差距的方式。如本节所述，这些限制与执行硬件组件的 MAD 触发器的灵活性直接对比。适当的结构合成方法必须能够利用固定语义流水线的 IP 核实现配置为 C_b 的 MADF 触发器，如式（11-10），其中 X、T、S 代表变量表达式的符号维度、阈值和每个端口的流处理数。第 11.5 节将概述这如何实现。

$$C_b = \{T_b \quad X_b \quad S_b\} \tag{11-10}$$

11.5　MADF 算法的流水线核心设计

一组 MADF 触发器转换为一组虚拟处理器（Virtual Processor，VP）执行，如图 11-22 中所述。有两个主要执行任务需要解决：即 VP 节点的生成和 FIFO 网络的互连。一个 VP 由三部分组成：

1）控制和通信封装（Control and Communications Wrapper，CCW）；

2）功能引擎（Functional Engine，FE）；

3）参数库（Parameter Bank，PB）。

图 11-22 VP体系结构

CCW实现了通过转换数据流可共享VP（代表MADF触发器）出入FE。这里的读取单位还实现了MADF网络必需的边缘FIFO缓冲。FE实现节点的功能和单位的核心部分。它可以是任何结构，但在这里它是一个高吞吐量的流水线核。PB为核的运行时间常数提供本地数据存储，如FIR滤波器抽头权重。VP的所有部分都是自动生成的；然而，流水线核心部分的灵活性供跨多个应用程序和MADF触发器配置重复使用，这只需创建和重用中一种基于核心的设计策略即可。自动生成有效产生FIFO缓冲器和控制器的专用数据流硬件是一个研究充分的领域（例如Dalcolmo等1998，Harriss等2002，Jung和Ha 2004，Williamson和Lee 1996）。本章的其余部分便与可重用FE的设计有关。

当一个VP的FE部分是流水线核，它被设计成白盒组件（White Box Component，WBC），并在第9章（Yi和Woods 2006）有描述；结构和行为在这里被描述为使用2阶FIR滤波器WBC例子，如图11-23所示。WBC由计算部分和状态空间组成，如第9章所述，但设计可重用的关键方面，即可配置核心，在于空间

图 11-23 2阶滤波器的WBC状态

部分恰当的电路设计。SFG 结构合成过程产生一个 MADF 触发器的基础数据状态空间。然后经一系列的扩增系统给 WBC 一个灵活的内部结构，无需重新设计，即可满足 MADF 经常变化的配置。

11.5.1　MADF 可配置流水线专用硬件结构的合成

SFG 结构合成导致流水线 WBC 的结构，仅仅是原始 SFG 算法的一个实时版本，其中的计算资源必须在输入流数组的 n 个元素之间有效地时分复用，整个 SFG 在交叉处理的情况下为单周期，在模块处理的情况下多周期。

若要启用交叉处理，则 WBC 状态空间增加过程的第一阶段是横向时延缩放。若要启用交叉共享，则 SFG 结构是 k 放缓的（Parhi 1999），SFG 结构合成造成的时延长度由系数 k 来衡量，如果是 n 输入流交叉处理，则 $k = n$。这种类型的操作被称为横向时延缩放。

如果需要模块处理，则基本符号从单个端口数组元素使用/产生的为持续的周期数。因此，对于 S 流，VP 状态空间应该有足够的能力激活与单个流相关的状态空间，并在为下一流加载状态空间前，处理任意数量的符号。这种负荷—计算—存储的行为是最适合于执行分布式内存内容的一个组成部分，由控制器调度确定活动的内存位置。这就是所谓的纵向时延缩放，每个 SFG 时延缩放为 S 元的 DisRAM。

有横向和纵向时延伸缩性生成方法，可重用 WBC 核的结构合成过程为允许触发器和流水线核配置，得到式（11-10）的系数，一个 4 阶结构合成已在开发。

（1）执行 MASDF 触发器 SFG 结构合成。选择 MADF 触发器 C，C_b是固定的专用基本配置。这被转换为 SFG 结构合成。MADF 触发器 C_b是最小的可能配置值，由此产生流水线架构，基本处理器 P_b可用，但因为参数化结构的经常变化，处理器可以执行整数超集配置。基本配置值越低，组件可以实现更高阶配置的范围越大。高阶配置执行越有效，C_b值越高。若两级 FIR 的 $C_b = \{1\ 1\ 1\}$，则 WBC 中 P_b 的 WBC 如图 11-23 所示。

（2）交叉处理的横向时延可伸缩性。要实现可变交叉操作的 k 放缓，所有时延的长度必须用常数 Q 衡量。所有最低级组件（加法器/乘法器）由预先设计的有固定流水线深度的核（图 11-23 中为同一值）建立，设计者是无法改变的。若要启用这些时延的缩放，还应加强输出时延来缩放单个流水线阶段的长度 Q。图 11-23 的流水线 FIR 生成的 FIR 电路结构如图 11-24a 所示。标记（Q）D 引用了维度（1，Q）的时延数组。需注意，所有时延长度现在都是横向缩放比例系数 Q 输出的最低级别组件上的附加时延锁链。这种类型的操作适合于有效执行作为移位寄存器的长时间时延的 FPGA。

（3）模块处理的纵向时延可伸缩性。模块处理的电路时延被纵向比例系数 P 进行缩放，允许组合的交叉/模块处理。这会导致维度（P，Q）的拖延数组。当这被应用于图 11-24a 所示电路时，创建图 11-24b 所示的 FIR 电路结构。注意在所有时延阵列上的纵向比例系数。这种基于 RAM 的微型嵌入式行为非常适合 FPGA 实现，因为这些可以用可编程逻辑实现小型分布式的 RAM（DisRAM）。这些 DisRAM 有相同的时间形式作为简单的时延，并因此不会影响电路结构中的边缘权重。

（4）重新定时结构最小化横向时延可伸缩性。当 P_b 用来实现比 C_b 高得多的 MADF 触发器配置时，会导致很大的时延长度。为了尽量减少这些，将定时应用于增强处理器结构。

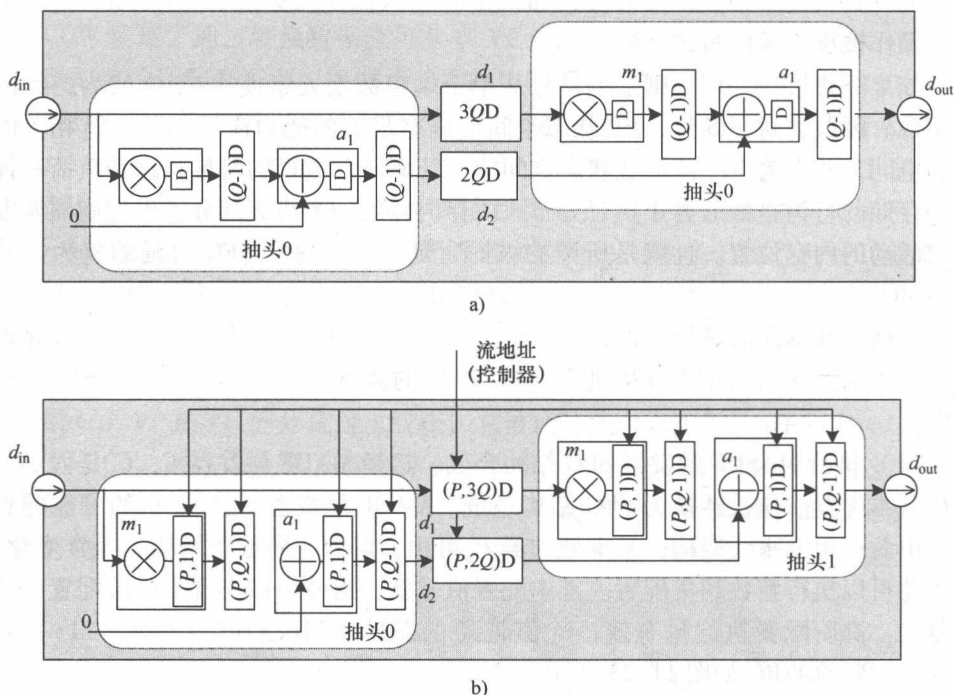


图 11-24 2 阶滤波器 WBC 的不同缩放

a）2 阶滤波器 WBC 横向缩放 b）2 阶滤波器 WBC 纵向缩放

11.5.2 WBC 配置

基于 WBC 结构 P_b 的配置创建后，必须用特定的 MADF 触发器配置使用。对配置 $P_b(C_b=\{T_b\,X_b\,S_b\})$ 与流水线周期 α_c（Parhi 1999），实现高阶 MADF 触发器 P，其中 X 是第 i 个维度的一个大小为 $x(i)$ 的 n 维符号，使用过程如下。

1）由式（11-11）确定横向缩放系数 Q。

$$Q = \left[\frac{1}{\alpha_c}\prod_{i=0}^{n-1}\frac{xi}{x_b i}\right] \tag{11-11}$$

2）用系数 Q 缩放所有边缘时延。缩放原始输出时延到长度$(Q-1)\times L$，L 是原流水线阶段的数目。

3）所有纵向时延的纵向缩放系数 P 由式（11-12）给出。

$$P = \frac{S}{S_b} \tag{11-12}$$

11.6　专用硬件网络的系统级设计与开发

本节将概述两个例子，即归一化格型滤波器（Normalized Lattice Filter，NLF）和固定波束形成器，以表明上一节中所述的特点。

11.6.1　设计示例：NLF

为了表明此架构的合成和开发方法的有效性，它被应用于 8 阶的 NLF 设计问题。MADF 图如图11-25所示，NLF 触发器作为本设计示例在专用硬件中实现的唯一部分。

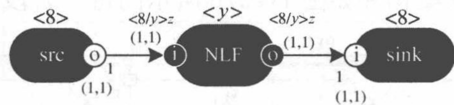


图 11-25　8 阶 NLF 的 MADF 图

如图 11-25 所示，src 和 sink 阵列生成记由 NLF 数组处理的 8 个标量符号数组。设计者通过控制画布上 NLF 触发器的变量 y 来控制 NLF 触发器数组的大小。反之，就确定了 NLF 触发器数组每个元素端口数组的大小 n。若要测试此 MADF 的合成及开发方法的有效性，则 P_b的 SFG 结构合成能力仅限于重定时（即提前进行结构开发，如在折叠/展开之前），实现优化的重点完全放在 MADF 的设计和探索能力上。P_b 对 $C_b=\{1b\ \ 1b\ \ 1\}$ 的标量符号的操作，通过最大化配置数目实现 SFO 灵活性最大化。目标设备是 Virtex－II Pro™族可实现的最小的可能成员。这使有效合成两个目标设备具有具体设计规则：

1）当$(P,\ Q)=(1,\ 1)$时，D_{type} = FDE

2）当 $P>1$ 时，D_{type} = DisRAM（RAM16x1s），其他 D_{type} = SRL16 + FDE

$C_b=\{1b\ \ 1b\ \ 1\}$ 的 8 阶 NLF 的 SFG 如图 11-26a 所示，NLF 阶段的 SFG 如图 11-27a 所示。如果它的结构拟建的最低级组件（加法器和乘法器）实施使用单级流水线的黑箱组件（当今的 FPGA 经常如此），那么 NLF 结构的一个特点是结构中的 36 递归循环，当两个流水线阶段相连时，重要的循环发生。对于单级流水线的加法器和乘法器，有流水线周期，（α），有 4 个时钟周期。因此由式（11-11），$Q=\dfrac{x_i}{4\times x_b}$。

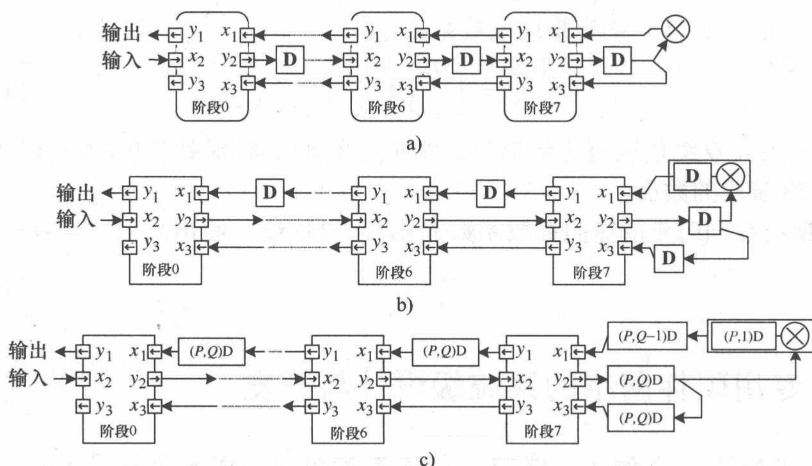


图 11-26　NLF 的 SFG、流水线结构和 WBC

a）8 阶 NLF 的 SFG　b）流水线 NLF 结构　c）NLF 的 WBC

基本处理器 P_b 通过分层 SFG 结构合成创建，并生成如图 11-26b 所示的流水线架构和如图 11-27b 所示的每个阶段的结构。横向和纵向的时延缩放和重新

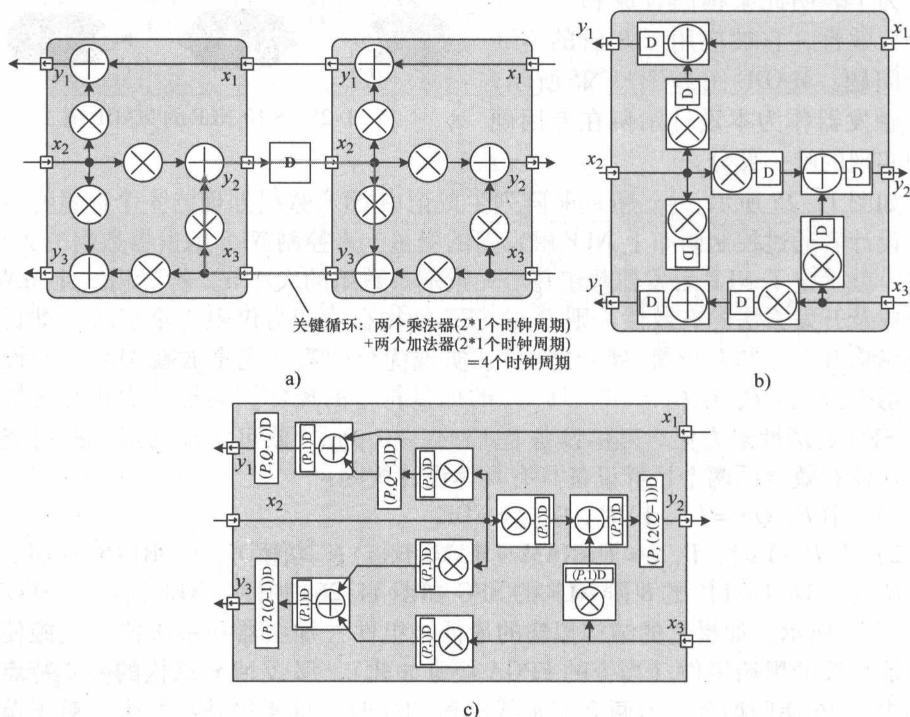


图 11-27　多阶 NLF 的 SFG、流水线结构和 WBC

a）多阶 NLF 的 SFG　b）流水线多阶 NLF　c）多阶 NLF 的 WBC

定时后，NLF 和 WBC 阶段结构如图 11-26c 和图 11-27c 所示。

3 个不同的 γ 值对给定结构的合成不同。$^8_1$BS - NLF，$^2_4$BS - NLF 和$^1_8$BL - NLF 是 γ 分别为 8，2 和 1 时的生成结构，每个 VP 在数据流对撞时交叉共享处理一个 68 元矢量（$^{68}_1$ BS - NLF），结果也用来说明 WBC 结构的灵活性。4 元矢量符号（$^1_4$BS - NLF_{16}）的 16 流模块处理示例也在表 11-4 中引用。这些结果说明了核心形成与高级结构开发方法的有效性。通过权衡族的触发器数目、每个触发器的符号大小和 MASDF 触发器循环调度的功能数目，转化 MASDF 规格，达到无需重新设计的有效优化方法。

表 11-4　Virtex - Ⅱ Pro　FPGA　上的 NLF 发送位置和路径合成结果

配置	片	逻辑				18 位乘法	吞吐量（M 样本/s）
		LUT	SRL	DisRAM	FDE		
$^8_1$BS - NLF	2784	1472			4840	312	397.4
$^2_4$BS - NLF	670	368			1210	78	377.9
$^1_8$BS - NLF	442	186	207		801	39	208.6
$^{68}_1$BS - NLF	442	186	207		801	39	208.6
$^1_4$BS - NLF	508	188	7	576	105	39	135.8

实施初期（$\gamma=8$，$^8_1$BS - NLF）创建一个 8 元素 VP 数组。鉴于大量的乘法器（mult18）执行所需要的最小设备可以是 XCV2VP70。不过，鉴于最初的 WBC 结构在流水线期间无效，因此把 γ 减少到 2 产生两个 4 元矢量处理器（$^2_4$BS - NLF）以几乎相同的吞吐量，使所需硬件资源的显著减少。这相当于每个 VP 无需额外硬件时，吞吐量增加了 3.9 倍。所需嵌入式乘法器数目的大量减少还允许在小得多的 XC2VP20 设备上执行。减小 γ 仍离 1 很远，产生单独的 8 元素矢处理器（$^1_8$BS - NLF）。减少了吞吐量，很大程度上节省了硬件。现在，NLF 数组可以在更小的 XC2VP7 设备上实现。

此示例展示了 MADF 合成方法可以通过简单的系统级设计空间开发来达到很好的执行结果。使用单独的流水线核心，这种方法已启用高度有效的结构（一对一映射有效率的 3.9 倍），很容易生成，且比 SFG 结构合成更简单且更连贯。此外，通过操纵单个 DFG 级参数，本设计示例可以在变化多端的实施要求下自动生成应用，使设备的复杂性数量级减少。这说明了作为一个系统级方法，基于核心的设计流与高度有效的执行结果和快速设计空间开发能力。

11.6.2　设计示例：FBF 系统

波束形成为雷达、声纳、生物医学和通信应用的空间滤波提供了有效和灵活的方法（Haykin 1986）。波束形成器通常用于被放置在不同的位置的传感器阵

列，以便它们能够通过接收传播波场采样“听”到接收信号。固定波束形成器（Fixed Beamformer，FBF）的结构如图 11-28 所示。

FBF 样品对 N 个传感器的复杂数据对撞，并对输出的 N 个样品逐个执行独立 FIR 滤波和缩放，引导空间对特定目标的波束形成器响应常数加权矩阵向量的各个元素。缩放值都相加。作为目标，当该波束形成器有靶向时，波束形成器的平均输出功率最大化。FBF 系统的结构非常有规律，并可以利用 MADF 建模域提供广泛而有效的设计空间开发。MADF 的 FBF 算法如图 11-29 所示。

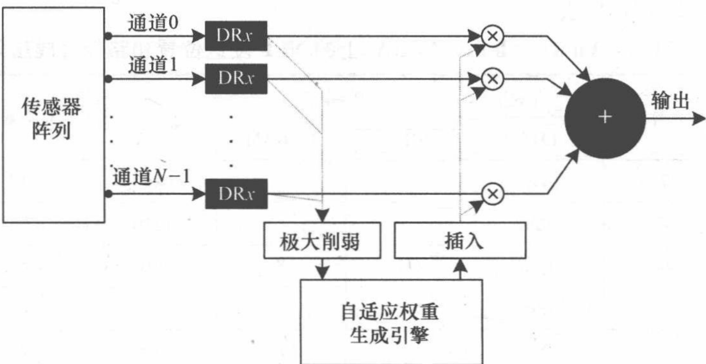


图 11-28　固定波束形成器概况



图 11-29　固定波束形成器的 MADF 图

MADF 图由一组 N 输入数组组成，每组用于一个传感器。这与 DRx 和 mult$K$ 触发器族的成员，以及总触发器（再次以黑色表示端口族）的端口族 i 的大小密切相关。因此，通过改变 N 的值，算法结构的参数化控制被用于可变数量的传感器。算法结构紧密耦合的实现结构，可紧密控制应用中 DRx 和 mult$K$ 核的数量。

此设计示例目的在于，$N=128$ 且面向 Xilinx Virtex II Pro 100 FPGA 的设计过程。核心库仅由复杂的乘法器、加法器和总核组成，因此整个系统就是由这些组成的。DRx 滤波器的长度为 32 抽头。已知此结构需要 16896 个乘法器，期望适应目标设备（其中只有 444 个可用）上提供的 18 位乘法器，这提出了一个高度资源约束下的设计问题。在这里提供两个方法来帮助解决这一问题：功能已经实现的结构操作和每个 VP 的多数据流处理。

若要探索应用中每个内核处理的通道数量，则每个触发器可处理 MADF 算

法的多个通道。这可使用图 11-30 所示的 MADF 结构。此处介绍的第 2 个参数 M，表示用于处理 N 通道数据的触发器数目。注意，DRx 和 mult$K$ 触发器上的端口都是大小为 M 的族，来表示以循环方式共享 N/M 数据流的处理（Mc Allister 等 2006）。对于合成，种类繁多的合成选择可用于选定设备上的 FBF 专用硬件系统，其广泛的实时性能和所需资源汇总在表 11-5。可编程逻辑（LUT/FDE）与 VP 功能（WBC，PB 和 CCW）的比例细目见表 11-6。

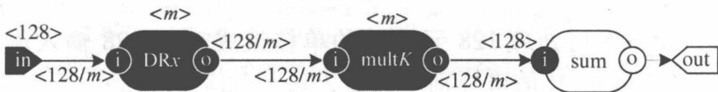


图 11-30　固定波束形成器的 MADF 图

表 11-5　Virtex - Ⅱ Pro FPGA 上的 FBF 发送位置和路径合成结果

$M(i)$	逻辑 LUT/SRL/DisRAM/FDE	18 位乘法 (%)	MUX F5/F6/F7/F8(%)	吞吐量 (M 样本/s)
1(i)	3493/16128/8448/5790 (8)/(37)/(19)/(6)	99 (22)	528/256/128/64 (1)/(1)/(1)/(1)	1.45
2(i)	4813/16128/8448/10844 (11)/(37)/(19)/(11)	198 (45)	512/256/128/64 (1)/(1)/(1)/(1)	3.18
4(i)	8544/16128/8448/21576 (19)/(37)/(19)/(22)	396 (89)	528/256/128/64 (1)/(1)/(1)/(1)	6.19
1(b)	3490/0/24576/5278 (8)/(0)/(56)/(5)	99 (22)	528/256/128/64 (1)/(1)/(1)/(1)	1.45
2(b)	4812/0/24576/8892 (11)/(0)/(56)/(9)	198 (45)	528/256/128/64 (1)/(1)/(1)/(1)	3.51
4(b)	8554/0/24576/17672 (19)/(0)/(56)/(18)	396 (89)	528/256/128/64 (1)/(1)/(1)/(1)	1.45

表 11-6　FBF 实现源分类

$M(i)$	LUT			FDE		
	% WBC	% CCW	% PB	% WBC	% CCW	% PB
1(i)	3.7	31.3	6.5	1.3	0	98.7
2(i)	3.6	28.7	69.5	0.9	0	99.1
4(i)	3.2	25.5	71.3	0.3	0	99.7

（续）

M (i)	LUT			FDE		
	% WBC	% CCW	% PB	% WBC	% CCW	% PB
1 (b)	3.7	31.3	6.5	1.3	0	98.7
2 (b)	3.6	28.7	69.5	1.0	0	99.0
4 (b)	3.2	25.5	71.3		0.3	99.7

初期实现由一个可处理 128 元向量的单核组成（即 128 输入流之间交叉共享）；增加为 2 和 4 的 M 值可分别增加 2.2 和 4.3 倍吞吐量；应该指出的是，用于多流共享的结构表现出了最少的资源差异。这是靶可移植性的核心结构抽象化的直接结果。同时在 LUT（如 16 位移位寄存器）或 VP 中，WBC 封装的 DisRAM 的费用昂贵（高达开销的 35%），其主要部分完全需要存储芯片上滤波器抽头和 SFO 参数库中的乘法器权重。如果不利用片上嵌入式 Block RAM，则这种存储弊端是不可避免。此外，M 值增加，由于抽头权重的数目与 M 无关，因此开销水平降低。CCW 带来少量 LUT 开销，转而开发嵌入式 MUX F5、MUX F6、MUX F7 和 MUX F8 的 FPGA MUX 来实现转换。这些都不在除设计以外的地方使用，因此很充裕。最后，应该指出的是在系统中的所有核心都 100% 地利用，取决于输入数据。

11.7 总结

本章描述了 FPGA 专用硬件网络的系统级设计和优化的可行技术。需要重视的是，在工业系统设计流中，专用的硬件一般会形成嵌入式 DSP 系统的一小部分，因此它的设计、集成和重用应妥善借用给工业异构系统设计流。

通俗地说，快速 DSP 系统实现方法都基于数据流中心。虽然流水线 DSP 核的快速执行本身就是数据流设计流量，但步骤的最终产品的集成、重用、操作，以及系统的软件部分是刚性的，不灵活的。随着方法的日益普及，这种情况一定会被纠正。

本章概述了大量的技术，来连通系统设计师所需要的和专用的硬件所允许的缺口。为产生专用硬件，接纳灵活性势在必行，本节展示可有效地合并到异构系统设计流中的专用硬件的创建、操作和重用。

MADF 作为一种 DSP 系统的建模方法，帮助封装了 DSP 系统灵活性所需的方面，特别是利用数据级并行性和控制如何影响实施方面的能力。这已被证实是一种有效的方法；如一个 NLF 滤波器设计示例，在设计方法上取得了令人印象深刻的成就。这包括通过简单的 DFG 级转换，增加的近乎 4 倍的执行效率，否

定了对复杂 SFG 操作的需要。此外，只通过从图级操纵单个参数，生产不同吞吐量和不同物理资源需求的 NLF 应用（在设备复杂性的数量级变化），这种方法的快速设计空间开发已证明有效。另外，在 FBF 设计示例中，启用快速设计空间开发，通过单个 DFG 参数操作生产各种专用设备，证明了此方法的有效性。

参考文献

Bhattacharyya S, Murthy P and Lee E (1999) Synthesis of embedded software from synchronous dataflow specifications. *Journal of VLSI Signal Processing* 21(2), 151–166.

Bilsen G, Engels M, Lauwereins R and Peperstraete J (1996) Cyclo-static dataflow. *IEEE. Trans Signal Processing* 44(2), 397–408.

Dalcolmo J, Lauwereins R and Ade M (1998) Code generation of data dominated DSP applications for FPGA targets *Proc. 9th International Workshop on Rapid System Prototyping*, pp. 162–167.

Gajski D, Vahid F, Narayan S and Gong J (1994) *Specification and Design of Embedded Systems*. Prentice Hall.

Grötker T, Liao S, Swan S and Martin G (2002) *System Design with System C*. Kluwer.

Harriss T, Walke R, Kienhuis B and Deprettere EF (2002) Compilation from matlab to process networks realised in fpga. *Design Automation for Embedded Systems* 7(4), 385–403.

Haykin S (1986) *Adaptive Filter Theory*. Prentice Hall, Englewood Cliffs, NJ.

Jung H and Ha S (2004) Hardware synthesis from coarse-grained dataflow specification for fast HW/SW cosynthesis *Proc. International Conference on Hardware/Software Codesign and System Synthesis*, pp. 242–29.

Kahn G (1974) The Semantics of a Simple Language for Parallel Programming *Proc. IFIP Congress*, pp. 471–475.

Kalavade A and Lee E (1997) The extended partitioning problem: hardware/software mapping, scheduling and implementation-bin selection. *Design Automation for Embedded Systems* 2(2), 125–163.

Keutzer K, Malik S, Richard Newton S, Rabaey JM and Sangiovanni-Vincentelli A (2000) System level design: orthogonolization of concerns and platform-based design. *IEEE Trans. CAD* 19, 1523–1543.

Lee E (1991) Consistency in dataflow graphs. *IEEE Trans. Parallel and Distributed Systems* 2(2), 2233–2235.

Lee E and Messerschmitt D (1987a) Static scheduling of synchronous data flow programs for digital signal processing. *IEEE Trans. Computers* C-36(1), 24–35.

Lee E and Messerschmitt D (1987b) Synchronous data flow. *Proc. IEEE* 75(9), 1235–1245.

Lee E and Sangiovanni-Vincentelli A (1998) A framework for comparing models of computation. *IEEE. Trans. CAD* 17(12), 1217–1229.

Lee EA (1993a) Multidimensional streams rooted in dataflow. *IFIP Transactions; Vol. A-23: Proceedings of the IFIP WG10.3. Working Conference on Architectures and Compilation Techniques for Fine and Medium Grain Parallelism*, pp. 295–306.

Lee EA (1993b) Representing and exploiting data parallelism using multidimensional dataflow diagrams *Proc. 1993 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP-93)*, pp. 27–30.

Lee EA and Parks TM (1995) Dataflow process networks. *Proc. IEEE* 85(3), 773–799.

McAllister J, Woods R, Walke R and Reilly D (2006) Multidimensional DSP core synthesis For FPGA. *Journal of VLSI Signal Processing* 43(2/3), 207–221.

Murthy PK and Lee EA (2002) Multidimensional synchronous dataflow. *IEEE Trans. Signal Processing* 50(8), 20647–20799.

Najjar WA, Lee EA and Gao GR (1999) Advances in the dataflow computational model. *Parallel Computing* 25(4), 1907–1929.

Parhi KK (1999) *VLSI digital signal processing systems : design and implementation*. John Wiley and Sons, Inc., New York.

Parks T, Pino J and Lee E (1995) A comparison of syncronous and cyclo-Static dataflow. *Conference Record of*

the 29th Asilomar Conference on Signals, Systems and Computers, pp. 204–210.

Rowson JA and Sangiovanni-Vincentelli A (1997) Interface-based design *Proc. 34th Design Automation Conference*, pp. 178–183.

Sriram S and Bhattacharyya S (2000) *Embedded Multiprocessors: Scheduling and Synchronization*. Marcel Dekker.

VSIA (2007) Vsia quality ip metric. Web publication downloadable from http://www.vsia.org/documents/.

Williamson M and Lee E (1996) Synthesis of parallel hardware implementations from synchronous dataflow graph specifications *Proc. 30th Asilomar Conference on Systems, Signals and Computers*, pp. 1340–1343.

Xilinx Inc. (2005) Virtex-ii pro and virtex-ii pro x platform fpgas: Complete data sheet. Web publication downloadable from http://www.xilinx.com.

Yi Y and Woods R (2006) Hierarchical synthesis of complex dsp functions using iris. *IEE Trans. Computer Aided Design* 25(5), 806–820.

第12章　自适应波束形成器实例

第8章包含了用基于SFG的DSP系统来创建高效的电路架构。在此描述中，很显然，许多实现可以用一种通用的，具有流水线级作控制参数的方式被创建出来。第10章指出，如果可参数化的这个特点在此参数范围里可用于高效的实现，那么这种通用架构的开发能在很大范围里得到应用。这只能通过识别DSP核心功能的关键参数得以实现，然后灵巧地派生出一个允许这些参数被改变的可伸缩架构，同时仍然可以保持线性的伸缩性能。这要求高级设计优化与第8章中强调的技术结合来使用。

本章主要讲述自适应波束形成的一个基于QR的IP核的开发。这个例子涵盖了大量的过程阶段，包括从数学算法的开发到一个可伸缩架构的设计。焦点是用于抓取原始架构的技术，然后将构架映射并折叠，得到一个高效并且可伸缩的实现来满足系统的要求。涵盖了可参数化的定时和控制管理的问题，这涉及早期的技术。

本章由以下部分组成。12.1节将给出一个自适应波束的介绍并且将在12.2节涵盖通用设计的过程。12.3节将突出自适应波束的应用并且12.4节将涉及基于QR的算法发展的内容。12.5节中包含程序构架的算法并且在12.6节将给出适用于实现的高效架构算法。一系列不同的架构被优化，然后这些构架被用于优化在12.7节中给出的通用架构。在12.8节将给出架构被重定时的细节，以及处理器协作的细节，因此在12.9节将突出可参数化的QR架构的实现。12.10节将探讨这个架构的通用控制问题。最后，一个波束应用在12.11节得到描述而且接下来作了总结。

12.1　引言

自适应波束形成是一种滤波的形式，从天线阵列中接收输入信号，多个空间上分离的天线，被称作为一个天线阵列。通常它的功能是抑制信号，在波束图中的干扰方向上，通过引入零陷来抑制除了想要“接收方向”的其他每一个方向上的信号。波束形成器的输出是天线阵列输入信号的一个加权线性结合，它由复杂的数字方程来表示，由于传入的数据有空间要素，因此可得到在幅度和相位中的最优值。

图12-1所示为用一个主天线和大量附属天线阐述的例子。主要的信号构成

是来自有着高度指向性的主天线的输入。附属的信号包含了干扰的样值，可能会淹没掉想要的信号。滤波器通过移除与主要输入信号一样的信号来消除此类干扰，来自附属和主要天线的输入数据被反馈到自适应滤波器中，如图 12-1 所示，从中进行权重计算。这些权重如图 12-1 所示被应用于时延的输入数据上来产生输出波束。这是本章重点，即一个自适应计算权重算法的选择和发展。

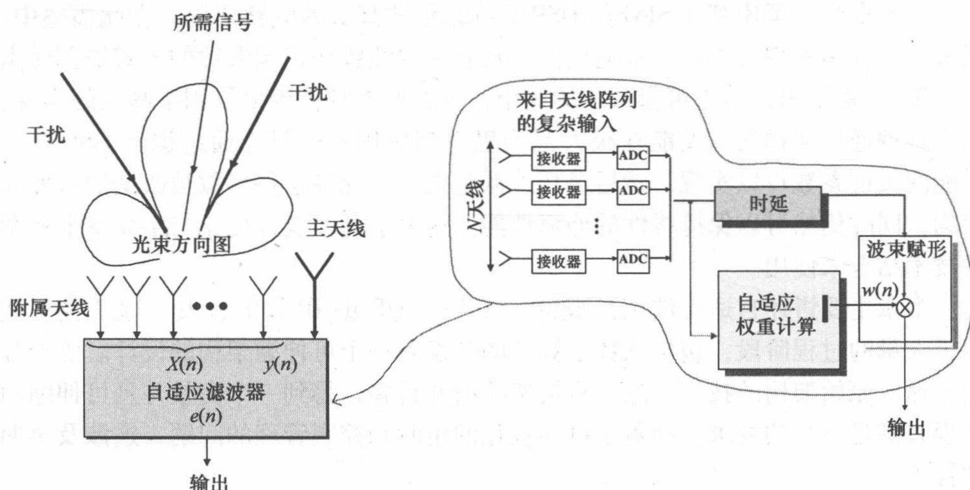


图 12-1 一个消除干扰的自适应波束形成器的框图。摘自 Lightbody *et al.*，© 2003 IEEE

有从军用雷达应用到通信、医学应用都可以使用的自适应波束形成器应用（Athanasiadis等 2005，Baxter 和 McWhirter 2003，Choi 和 Shim 2000，de Lathauwer等 2000，Hudson 1985，Wiltgen 2007）。由于这样潜在的一个核心应用，本章将通过调查一个 IP 核的发展来演示这种大量的自适应波束形成器应用中关键计算的建立。

12.2 通用设计过程

在一个新硬件的核心发展中，会跟随着一系列的设计阶段。图 12-2 所示为一个被应用于一次性实现的发展中的典型设计进程的概述。它也给出了在通用 IP 核设计中要额外考虑的方面，正如第 10 章中突出的。大量的关键问题得到了处理。

这个进程从一个问题的详细说明和设计目的开始。这时候，会对使用的算法进行优化来开发一个通用的终端产品。这些优化包括在通用的核心发展中最原始的时间和金钱方面的额外开销，因此如果这个 IP 核在未来的算法中得到使用，则这个开销将得到更多的回报，这一点是肯定的。这个发展可适用于一系列的应

用吗？或者说仅仅是一个一次性的需求？

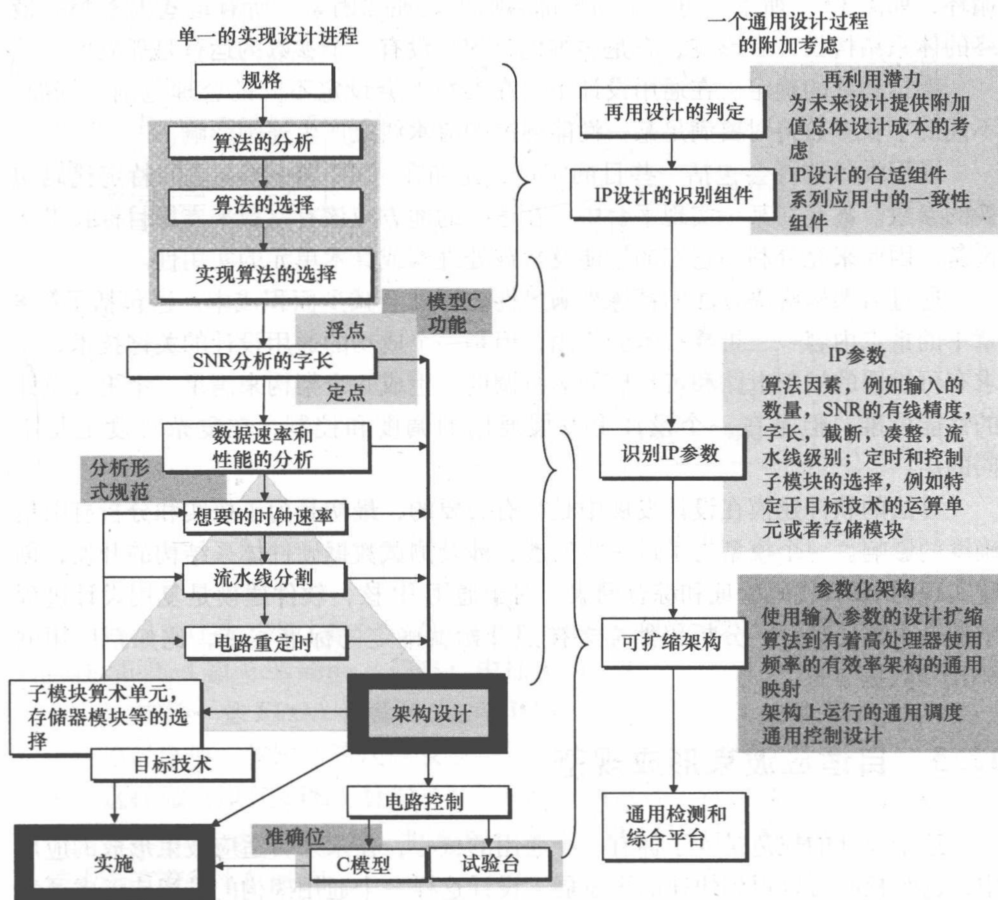


图 12-2　通用设计进程。摘自 Lightbody *et al.*，© 2003 IEEE

分析，然后来决定最合适的算法。算法的选择是关键，因为成功的硬件实现，要求详细理解在数学功能上是如何实现的。这会影响一个电路在尺寸、关键路径和功耗上的所有性能。

核心功能的识别。可能只有一个关键组件是适合作为 IP 核实现的。这些功能会从一个应用传递到另一个应用中，因此在未来的应用中决定预期的规格变化是很有必要的。是否所有变量在通用设计中都可用参数来定义？或者其他的技术会要求有一定的设计灵活性？

字长分析是设计中用来决定固定/浮点算法及设计中凑整和截断的参数。

架构设计细节会决定模块速率和尺寸。这将取决于目标技术或特定的 FPGA

设备，其会影响层次上并行性的分析模块和流水线的选择。它是一个相互关联的循环，如图12-2所示，每一个因素都影响了其他的因素。所有这些因素都对最终的体系结构设计有影响，它是多维优化的，没有一个参数的运行是孤立的。

架构参数的确定。在通用设计中，在参数上会设定不同的合理范围。例如，不同的字长参数将对要满足某一性能标准的流水线级产生连锁影响。

通用设计选择会包括一些目的实现，比如在ASIC和FPGA之间特定代码切换的参数。甚至在某一实现平台中，在适当的地方应该有参数来支撑目标技术和设备，因此来充分利用它们的性能及板载处理器或算术单元的实用性。

通过对架构解决方法的精炼来满足性能标准会减少面积成本。这包括了第8章中的重点内容——折叠技术的应用，但是一个成功的通用设计的关键技术，要求有可扩展的控制电路和可扩展的运行调度。生成一个架构来满足一个更大设计的性能标准，但是在一个设计上发展通用的调度和控制，在复杂程度上是不同的。

算法的软件建模在设计发展中是很有必要的，最初是为了核实和分析有限精确度的影响。一个模型为了进一步发展，涉及测试数据硬件体系结构的开发，创建实现的HDL代码验证和综合网表。对于通用IP核，软件建模是复用设计进程中的一个重要部分，分析仍然需要按照开始就确定的标准实施，比如新应用的SNR和数据字长。

12.3 自适应波束形成规范

这个设计的规范是为了拥有一个通用的核心，在未来自适应波束形成的应用中，这个核心可以得到快速的再发展。设计这样一个通用架构的进程应考虑开发时间，并且在决定应用重点方法之前应该考虑架构的未来潜力。但是，如果这样一个通用核心有复用的潜力，那么还是能得到收益的，并且会提供强大有用的设计库。

自适应波束形成是一种适用于一些范围内的通用算法，比如从信号的医学分离到军用雷达应用。通用设计开发的关键因素是，在某范围内自适应波束形成的应用程序的关键组成部分在一定程度上是一致的，这样才适合开发成一个IP核。为了扩展核心的潜力，就需要有处理不同种类问题的能力，比如以下问题。

1. 输入的数量

波束形成器的设计需要支持各种各样的附属输入和主要输入，如图12-3所示。因为输入数据是来自N天线的整个模块的输入数据，所以设计了计算权重的方法。(一般来说，虽然只需要输入数据的部分，但是应该至少需要$2N$的数据样值)。从相同的输入数据模块，到产生波束形成器输出的整体过程中，都需

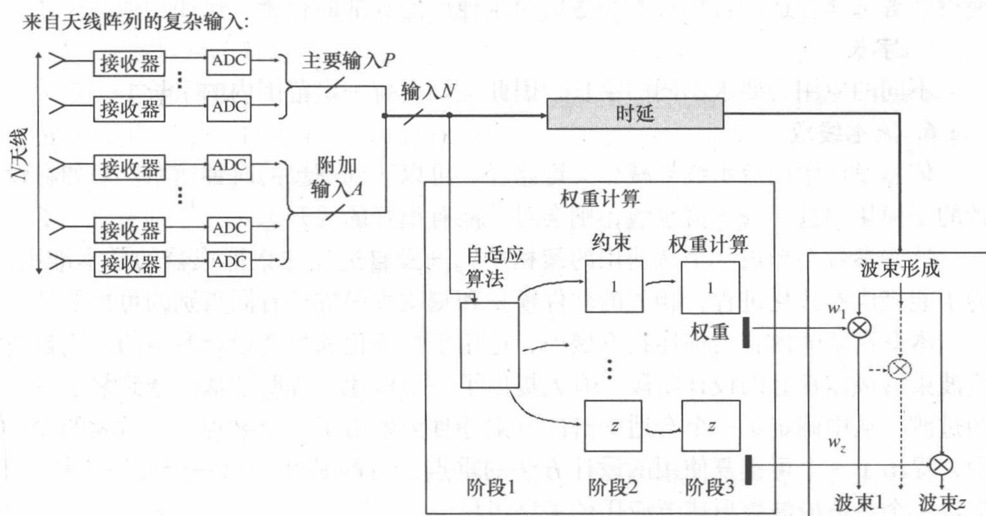


图 12-3 多波束自适应波束形成器系统。摘自 Lightbody *et al.*，© 2003 IEEE

要进行权重计算。对于最后的设计，一个更加高效的后台处理器得到了发展，比如在 Shelhed 和 McWhirter（1993）中描述的权重。

2. 支持一些 FPGA 设备和/或 ASIC 技术

通过添加一些额外的代码及参数，同样的核心设计能被重新赋予不同的技术。这样做可以使 FPGA 针对 ASIC 的原型所设计。这也要顾及低产量的实现。不敢保证 ASIC 设计的开销。同样的，形成 ASIC 铸造厂的快速的可再发展核心能力会有着很大的利润。

3. 支持一系列性能标准的能力

在自适应波束形成中的应用，创造了一个大跨度的变化所需的功能。对一些应用，比如移动通信、功率考虑及芯片面积，都可能是设备的驱动标准。对其他而言，一个高数据速率的系统能成为主要的目标。

4. 可扩缩架构

为了在支持一个大范围设计标准时拥有灵活性，一个可扩缩架构需要通过提高物理硬件的级别来匹配规范的需求。驱动可扩缩架构的一些关键点是：

1）想要的数据速率；

2）尺寸限制；

3）时钟速率限制；

4）功率限制。

系统要求的时钟速率是依靠架构设计和目标技术来实现的。具体的系统要求

使设计者选择合适的目标技术并考虑对其他性能标准的权衡，比如功率和尺寸。

5. 字长

不同的应用会要求不同的字长，因此应该支持一定范围内的字长。

6. 流水线级

依靠设计中的流水线来减少关键路径，可以达到理想的时钟速率。假如在设计的子模块中选择一个流水线，则会对性能有很好的提升。

这些参数会形成一个从通用的架构中，开发自适应波束形成器的基本准则。为了能完成参数化进程，相关的软件模型和测试程序应该有同级别的可扩展性。

本章剩余的内容会描述在发展中，适用于参数化和快速设计原型的通用自适应波束形成器核心的设计阶段。有人提出了一个概述，说明了从问题到数学算法的过渡。从中确定了一个合适的解决方案并且派生出了一个架构。在后续的设计中，提出了一个可重复使用的设计方法的重点，目的是开发出一个通用的核心，形成一个自适应波束形成器应用的关键组件。

12.4 算法的开发

一个自适应滤波器的目标是可以根据运行的环境持续优化自身。许多数学上的算法和相当复杂的算法，都是根据一个最优化标准来计算滤波器权重的。典型的，为了最小化误差函数。这个误差函数是理想性能和实际性能之间的差。如图 12-4 所示，展示了这个进程。

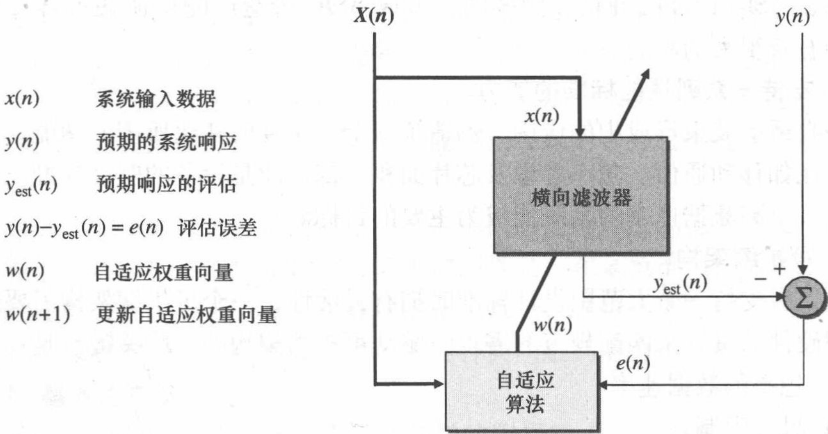


图 12-4 自适应滤波系统

大量的研究调查用不同的方法来计算滤波器权重。由 Haykin（2002）提出的自适应滤波器理论对自适应滤波器做出了一个全面的介绍。要求先详细地分析算法的复杂度和性能，再来决定一个合适的算法。但是在为一个具体应用找到最适宜的自适应算法的过程中并没有特别的技巧。选择合适的算法，归根结底是算法的特征范围的权衡，比如：

1）收敛速率，也就是自适应算法到达一个优化解决方案限度的速率；

2）稳态误差，即接近一个最佳的解决方法；

3）跟踪输入数据中的统计变量的能力；

4）计算的复杂度；

5）处理不良输入数据的能力；

6）在实现中使用字长变化的敏感度。

用于自适应滤波器的递归算法有两种方法，即 Wiener 滤波器理论和最小二次方理论（Least - Squares，LS），分别引出了在第 2 章中提到的 LMS 和 RLS 算法。这些算法选择的关键是算法的复杂度和性能。

12.4.1　自适应算法

相比于 LMS 解决方案，大家都更赞同 RLS 解决方案，可以在静止环境提供更加优秀的收敛速率。因此一个 RLS 方案在非平稳的环境下可能会有更快速的反应，但是这并不是一个直接简单的比较（Eleftheriou 和 Falconer 1986，Eweda 1998，Eweda 和 Macchi 1987，Haykin 2002，Kalouptsidis 和 Theodoridis 1993）。从这些参考文献中可以得知，一般着眼于以下几方面。

一般而言，在标准情况下 RLS 算法的跟踪性能和收敛速度比 LMS 算法更好，特别是在一种情况下，那就是在中高信噪比的情况下（Kalouptsidis 和 theodoridis 1993）。

LMS 算法的收敛速率取决于步长的选择，步长越大收敛速度越快。但是当步长过大时，收敛速率快并不能补偿在稳定状态下速率更慢时，算法平滑噪声的好处。这里有一些在最开始就考虑到较大步长的规范版本，但是用算法处理它的稳定状态时步长会被减小（Bitmead 和 Anderson 1980，Kalouptsidis 和 Theodorides 1993，Morgan 和 Kratzer 1996）。

RLS 算法的收敛速率不依赖于输入相关矩阵的特征值的传播。而 LMS 算法不是这样，因为当相关矩阵的特征值传播得很好时，我们宁愿收敛速率慢一点，（Eweda 和 Macchi 1987），也就是特征值有效地形成一个收敛时间常数，并确定常数的大小 μ，从而确保稳定性（Eleftheriou 和 Falconer 1986）。

自适应算法的存储器决定收敛速率。RLS 算法包含了一个遗忘因子 λ，它对较新的数据有着更重要的作用。它的值在 0 ~ 1 之间并且对决定一窗数据有所作

用，LS 方案就是在这个数据上实施的。伴随着较大的 λ 值，窗口长度会变长并且会占用一个更大的存储空间。λ 越小需要的存储空间越小，并且让算法能够在数据中跟踪统计的变化。但是，λ 也控制着收敛速率和稳态误差。在固定的环境中，λ 接近于 1 的慢适应性能可得到最好的稳态性能。相反，更小的 λ 的值会导致更快的收敛速率但是也会导致更大的稳态误差。这与存在于 LMS 算法中的内存收敛关系相似，这由步长 μ 决定。

优化的选择归根结底是收敛速率、稳态误差、跟踪能力、数值稳定性和计算复杂度的权衡。RLS 算法的主要缺点是它的计算复杂度，但是随着技术的发展，RLS 在实时应用中的使用变得可行。这里提出的例子中，选择 RLS 方案比选择 LMS 方案要多，这都归因于收敛速率的下降和对不良数据敏感度的下降。

12.4.2 RLS 实现

正如第 2.7.4 节中重点强调的，标准 RLS 算法要求相关矩阵的明确计算，$\boldsymbol{\phi}(n)=\boldsymbol{X}^{\mathrm{T}}(n)\boldsymbol{X}(n)$。这是一个密集型的计算，它对调整问题的条件数量是有影响的，这会为了达到有限字长系统的稳定性，对要求的字长造成负面影响。要得到一个更加稳定的方式可以使用权重计算，这就同时避免了相关矩阵的计算和逆用 QR 分解，QR 分解是一种具有良好数值特性的正交三角化。

为了实现自适应滤波器权重的计算，选择使用 QR 分解作为中心算法（Gentleman 和 Kung 1981，McWhirter 1983）。

12.4.3 通过 QR 分解求解 RLS

$P\times N$ 维的数据矩阵 $\boldsymbol{X}(n)$ 被分解为一个 $N\times N$ 维的上三角矩阵 $\boldsymbol{R}(n)$，通过一个单位矩阵 $\boldsymbol{Q}(n)$ 的应用，变成了

$$\boldsymbol{Q}(n)\boldsymbol{X}(n)=\begin{bmatrix}\boldsymbol{R}(n)\\ \boldsymbol{O}\end{bmatrix}\tag{12-1}$$

式中，O 是 0 矩阵，假设 $N<P$，由于 $\boldsymbol{Q}(n)$ 是一个单位矩阵，因此

$$\boldsymbol{\phi}(n)=\boldsymbol{X}^{\mathrm{T}}(n)\boldsymbol{X}(n)=\boldsymbol{X}^{\mathrm{T}}(n)\boldsymbol{Q}^{\mathrm{T}}(n)\boldsymbol{Q}(n)\boldsymbol{X}(n)=\boldsymbol{R}^{\mathrm{T}}(n)\boldsymbol{R}(n)\tag{12-2}$$

三角矩阵 $\boldsymbol{R}(n)$ 是数据相关矩阵的 Cholesky 因子 $\boldsymbol{\phi}(n)$。由于 $\boldsymbol{Q}(n)$ 是单位矩阵，因此原始系统等式表达为

$$\|\boldsymbol{J}(n)\|=\|\boldsymbol{Q}(n)\boldsymbol{e}(n)\|=\left\|\underbrace{\boldsymbol{Q}^{\mathrm{T}}(n)\boldsymbol{X}(n)}_{\boldsymbol{R}(n)}\boldsymbol{W}_{\mathrm{LS}}(n)+\underbrace{\boldsymbol{Q}^{\mathrm{T}}(n)\boldsymbol{y}(n)}_{\boldsymbol{u}(n)}\right\|\tag{12-3}$$

于是出现了最小平方权向量 $\boldsymbol{w}_{\mathrm{LS}}(n)$，必须满足等式

$$\boldsymbol{R}(n)\boldsymbol{w}_{\mathrm{LS}}(n)+\boldsymbol{u}(n)=0\tag{12-4}$$

由于 $\boldsymbol{R}(n)$ 是一个上三角矩阵，因此通过使用反置换，权重的问题能得到解决。QR 分解是这个 QR 因式分解的一个扩展，这使得当新数据进入数据矩阵时，矩阵能够被再次三角化，不必从原始的方阵格式中计算三角化。换而言之，当新数据进入时它更新了旧的三角矩阵。数据矩阵 $\boldsymbol{X}(n)$ 和测量向量 $\boldsymbol{y}(n)$ 在 n 时，通过先前结果的矩阵和向量以一种递归的方式被表示，并且新的数据如下：

$$\boldsymbol{X}(n) = \begin{bmatrix} \lambda(n)\boldsymbol{X}(n-1) \\ \underline{\boldsymbol{x}}^{\mathrm{T}}(n) \end{bmatrix} \tag{12-5}$$

及

$$\boldsymbol{y}(n) = \begin{bmatrix} \lambda(n)\boldsymbol{y}(n-1) \\ \underline{\boldsymbol{y}}(n) \end{bmatrix} \tag{12-6}$$

式中，$\underline{\boldsymbol{x}}^{\mathrm{T}}(n)$ 和 $\underline{\boldsymbol{y}}(n)$ 在 n 时刻形成了额外的行。算法的二次方根的形式得到如下实现：

$$\boldsymbol{Q}^{\mathrm{T}}\begin{bmatrix} \lambda^{0.5}\boldsymbol{R}(n-1) \\ \underline{\boldsymbol{x}}^{\mathrm{T}}(n) \end{bmatrix}\boldsymbol{W}_{\mathrm{LS}}(n) = \boldsymbol{Q}^{\mathrm{T}}(n)\begin{bmatrix} \lambda^{0.5}\boldsymbol{u}(n-1) \\ \underline{\boldsymbol{y}}(n) \end{bmatrix} + \boldsymbol{Q}^{\mathrm{T}}(n)\boldsymbol{e}(n) \tag{12-7}$$

其中，$\beta = \lambda^{0.5}$。然后给出了

$$\boldsymbol{Q}^{\mathrm{T}}\begin{bmatrix} \boldsymbol{\beta}(n)\boldsymbol{R}(n-1) & \boldsymbol{\beta}(n)\boldsymbol{u}(n-1) \\ \underline{\boldsymbol{x}}^{\mathrm{T}}(n) & \underline{\boldsymbol{y}}(n) \end{bmatrix} = \begin{bmatrix} \boldsymbol{R}(n) & \boldsymbol{u}(n) \\ \boldsymbol{O} & \boldsymbol{\alpha}(n) \end{bmatrix} \tag{12-8}$$

计算得出

$$\begin{bmatrix} \boldsymbol{R}(n) \\ \boldsymbol{O} \end{bmatrix}\boldsymbol{W}_{\mathrm{LS}}(n) = \begin{bmatrix} \boldsymbol{u}(n) \\ \boldsymbol{\alpha}(n) \end{bmatrix} \tag{12-9}$$

$\boldsymbol{\alpha}(n)$ 与后面的最小二乘的残留 $\boldsymbol{e}(n)$ 相关，在时刻 n 如下

$$\boldsymbol{e}(n) = \boldsymbol{\alpha}(n)\boldsymbol{\gamma}(n) \tag{12-10}$$

式中，在消除 $\underline{\boldsymbol{x}}^{\mathrm{T}}(n)$ 过程中产生余弦结果 $\boldsymbol{\gamma}(n)$。

RLS 的 QR 方案的高级关系图如图 12-5 所示。

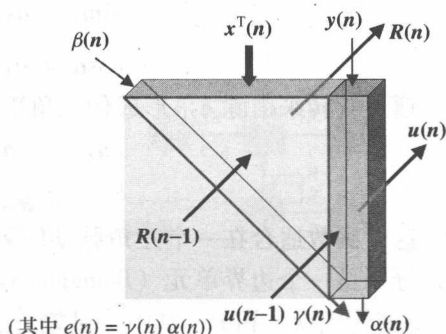


图 12-5　QR - RLS 方案的高级关系图

12.4.4 用于 QR 因数分解的 Givens 旋转

一系列数值上稳定且强大的 RLS 算法已经由一系列 QR 分解方法得到了改进，比如 Givens 旋转（Gentleman 和 Kung 1981，Givens 1958，McWhirter

1983)，CORDIC（Hamill 1995）及 Householder 变换（Cioffi 1990，Liu等 1990，1992，Rader 和 Steinhardt 1986)。Givens 旋转是矩形平面旋转，它用来消除一个矩阵中的元素。通过应用一系列的连续 Givens 旋转，一个矩阵能通过消除对角线之下的元素来被三角化。这个运算被称作 QR 因式分解，比如一个矩阵 $\boldsymbol{X}(n)$ 被分解为一个上三角矩阵 $\boldsymbol{R}(n)$ 和一个下三角矩阵 $\boldsymbol{Q}(n)$，如下：

$$\boldsymbol{X}(n) = \boldsymbol{Q}(n)\boldsymbol{R}(n) \tag{12-11}$$

$\boldsymbol{X}(n)$ 矩阵通过旋转矩阵被预乘。计算旋转参数，使第一列的次对角线元素为 0。然后再使下一列的次对角线元素为 0，以此类推，直到形成等价的上三角矩阵。

Givens 通过一系列旋转进行这个运算，它使用一个 2×3 矩阵的例子来进行描述，如下：

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{11} & a_{12} & a_{13} \end{bmatrix} \tag{12-12}$$

这个矩阵通过消除元素被转换为一个伪三角矩阵 a_{21}。这是通过乘以旋转矩阵来实现的

$$\begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix}$$

因此

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{11} & a_{12} & a_{13} \end{bmatrix}\begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & -\cos\alpha \end{bmatrix} =$$

$$\begin{bmatrix} a_{11}\cos\alpha + a_{21}\sin\alpha & a_{12}\cos\alpha + a_{22}\sin\alpha & a_{13}\cos\alpha + a_{23}\sin\alpha \\ -a_{11}\sin\alpha + a_{21}\cos\alpha & -a_{12}\sin\alpha + a_{22}\cos\alpha & -a_{13}\sin\alpha + a_{23}\cos\alpha \end{bmatrix}$$

为了消除 a_{21}，我们需要解决 $[-a_{11}\sin\alpha + a_{21}\cos\alpha] = 0$

因此，从三角法

$$\sin\alpha = a_{21}/\sqrt{a_{11}^2 + a_{21}^2}$$

$$\cos\alpha = a_{11}/\sqrt{a_{11}^2 + a_{21}^2}$$

通过旋转来消除 a_{21} 形成伪三角矩阵

$$\begin{bmatrix} a_{11\text{new}} & a_{12\text{new}} & a_{13\text{new}} \\ & a_{12\text{new}} & a_{13\text{new}} \end{bmatrix} \tag{12-13}$$

这个函数适合在一个三角脉动阵列上实现，如图 12-6 所示，包含了两类单元，分别是一个边界单元（Boundary Cell，BC）和一个内部单元（Internal Cell，IC)。旋转元素 x 和 R，其中 x 是输入到单元中的输入值，R 是那个单元的内存中保留的值。旋转参数 $\cos\alpha$ 和 $\sin\alpha$ 在 BC（指圆）中计算，输入到那个单元的 x

值被消除，并且在单元中的值 R 根据下一次迭代的旋转和存储被更新。旋转参数通过整行 IC（指正方形）保持不变，持续进行旋转。图 12-6 所示为消除 x_{21}，这与先前例子中说明的第一列次对角元素 a_{21} 有关。实际上，R 和 x 值被当作一个极坐标（R，x）。

消除 x 输入到 BC 中，可以通过 R 旋转 α 角度得到实现，如下：

$$R_{\text{new}} = R\cos\alpha + x\sin\alpha = \frac{R^2 + x^2}{\sqrt{R^2 + x^2}} = \sqrt{R^2 + x^2}$$

式中

$$c = \cos\alpha = \frac{R}{R_{\text{new}}}$$

并且

$$s = \sin\alpha = \frac{x}{R_{\text{new}}}$$

BC 中同样的旋转是遍及 IC 的：

$$R_{\text{new}} = cR + sx$$
$$x_{\text{new}} = cx - sR$$

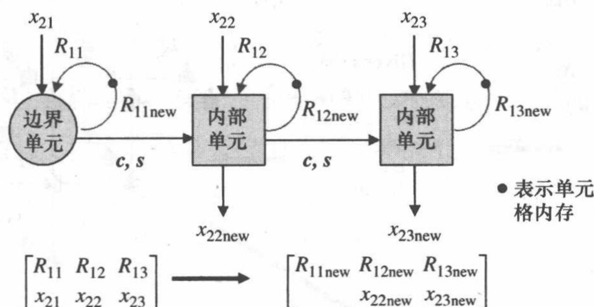


图 12-6　Givens 旋转

通过连续的 Givens 旋转，可以实现 QR 因式分解，形成一个 $n \times n$ 的矩阵。图 12-7 所示为一个 3×3 的矩阵例子。被流水线化后，就允许 R 的值被反馈到下一次迭代的单元中。

接下来的小节将对从一个数学算法发展成一个硬件架构的进程给予更详细的介绍，通过采用 Givens 旋转的 QR 分解来解决 RLS 算法的问题。

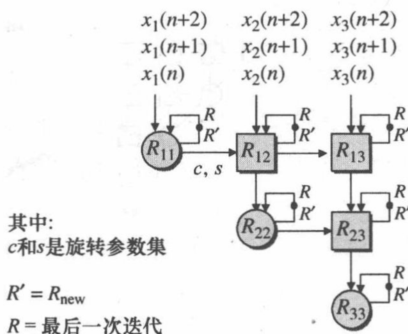


图 12-7　应用于矩阵的 QR 因式分解

12.5 从算法到结构

在先前的章节中，细节部分已经在自适应算法的基础上分析给出了，并且特别是 RLS 算法问题通过使用 Givens 旋转的 QR 分解得以解决。实现高性能电路的一个关键方面就是确保在硅硬件上算法的有效映射。这涉及一个硬件架构的优化，在这个架构中独立的运算都是并行形式，以便提高吞吐量。此外，流水线会在处理器模块中得到使用，来达到更大的吞吐量。同时使用并行形式和流水线的架构是一个脉动阵列。它的处理能力来自于大量简单单元的并行使用，而不是一些非常强劲的单元连续使用。结果就是局部互连的处理器是有规律地排列起来的。随着技术的进步，门时延已经不是控制电路性能的主要因素。反而，互连长度的影响比一个电路的关键路径和功耗的影响更大，因此保持局部的互连很重要。

图 12-8 所示为从算法到架构的进程，其框图中的起点是由 QR 分解解决的

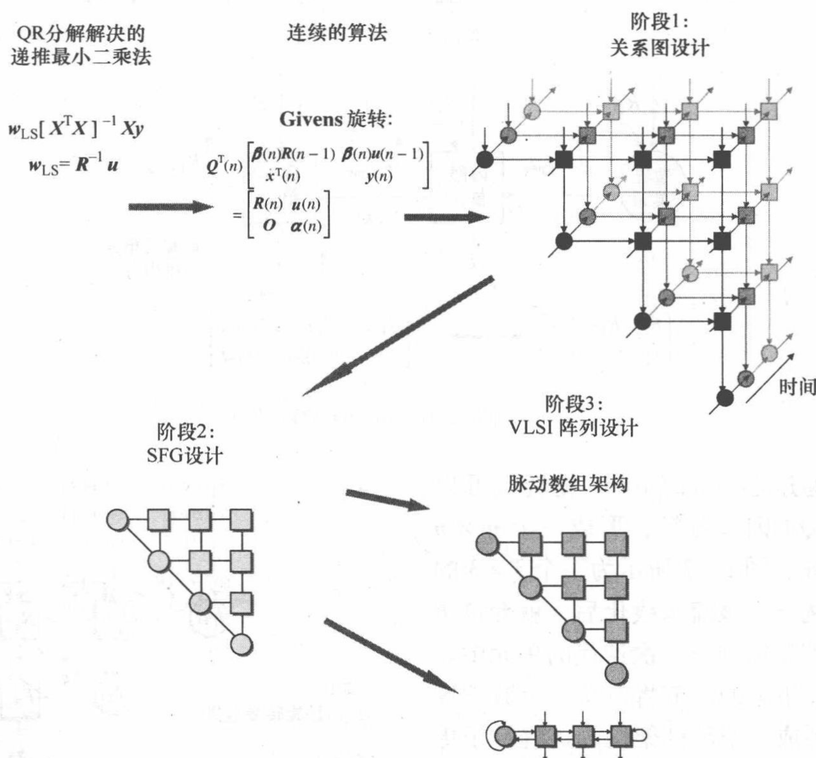


图 12-8　从算法到架构

RLS 算法。框图的下一个阶段描述了使用一个连续的通过 QR 分解解决的 RLS 算法。就是说，对算法的每个迭代，新设定的值被输入到等式中，因此可以被解决。QR 运算能被描述为一个运算的三角数组。在三角形的顶部输入数据矩阵，并且随着每行的其他项被消除，因此会得到一个上三角矩阵。图 12-8 所示 DG 描述了这个三角化的过程。这个框图中级联的三角数组表示了在时间上的迭代，也就是每个数组表示一个新的迭代。级联数组间的箭头突出了随时间变化的关系。

从 DG 中可以得到一个合适的 SFG，这表示一个构架可以开发。以下各节将简要介绍在图 12-8 中，这些阶段所描绘的每一个环节。

12.5.1　DG

数据间的依赖关系在一个 DG 中能被识别出来，这就允许通过将该算法分解为节点和箭头，来识别到并行的最大级。节点概述了计算过程并且箭头的方向说明了运算有依赖性。图 12-9 所示为三维 DG 的 QR 算法。框图显示了三个连续的 QR 迭代，用“依赖弧线”连接有依赖性的运算。一些可变标号为了清晰度被省略了。

新的数据通过 $\underline{\boldsymbol{x}}^{\mathrm{T}}(n)$ 表示。n 表示了算法的迭代次数。总之，QR 数组执行输入向量 $\underline{\boldsymbol{x}}^{\mathrm{T}}(n)$ 的旋转，值 R 被保存在 QR 单元的内存中，以便输入到 BC 的输入值 x 被旋转至 0。这样的旋转沿着经 QR 单元间的 IC 水平线延续出去。从这个 DG 中，可能推导出大量的 SFG 表达式。这里使用最明显的映射是沿时间（即 R）箭头映射 DG。

12.5.2　SFG

如图 12-10 所示，描述了从 DG 到 SFG 的转换。为了从 DG 中推导出 SFG，DG 的节点被分配到处理器，然后它们的运算被规范到这些处理器上。处理器分配的共同技术是所有相同节点的线性映射沿着一条直线传输到单个处理器上。这由映射向量 $\boldsymbol{d}$（见图 12-10）数值表示。线性调度被用于决定某个命令，这个命令是在处理器上运算的。图 12-10 中的调度线指示了运算位置，这个运算在每个周期中是并行执行的。数学上的，它们由正常的调度向量 $\boldsymbol{s}$ 到调度线表示，在方向上依赖操作，也就是它说明了一个命令中每一行运算的顺序。

这有两个基本规则，分别是控制映射和调度，来确保保持操作的顺序。给定一个 DG 和映射向量 $\boldsymbol{d}$，调度是允许的，如果只有：

1）所有依赖弧线流向同一方向跨过调度线；

2）调度线不能与映射向量 $\boldsymbol{d}$ 并行。

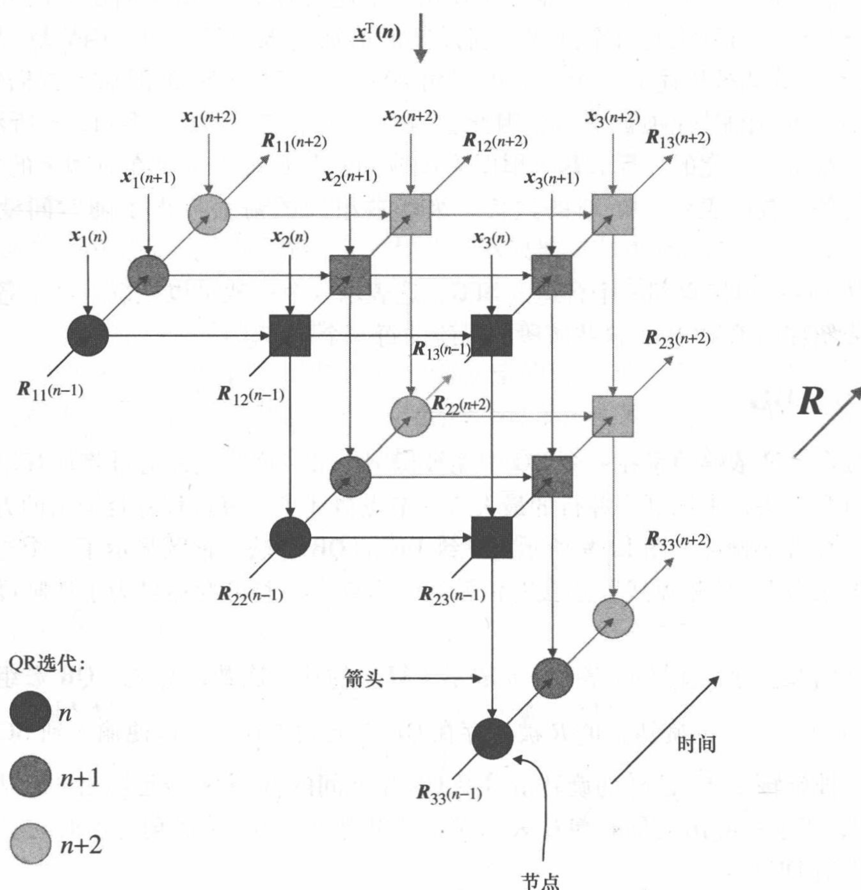


图 12-9　QR 分解的 DG

在 QR 例子中（见图 12-10），DG 表达式单元中的每个三角数组表示一个 QR 更新。当级联时，DG 表示连续的 QR 更新。通过沿时间轴映射，所有 QR 更新会被分配到一个三角 SFG 上，如图 12-10b 所示。

在 DG 中，*R* 值随时间轴经一个 QR 更新传到另一个 QR 中，通过级联的三角数组被表示出来。这个转换通过图 12-10b 中的环被表示得更加简洁，通过需要保持的算法时延，将 *R* 值返回到单元格中用于下一个 QR 更新。这被称为递归环。

简单的 SFG 是假设节点中执行的所有运算运行一个周期，与算法时延一样，由小的黑色节点表示。这些算法时延分割了算法的迭代并且是算法的一个必要部分。因此 SFG 是比 DG 更简洁的算法表达式。

本章剩余的部分给出了推导一个高效架构的进程及 SFG 算法表达式硬件实

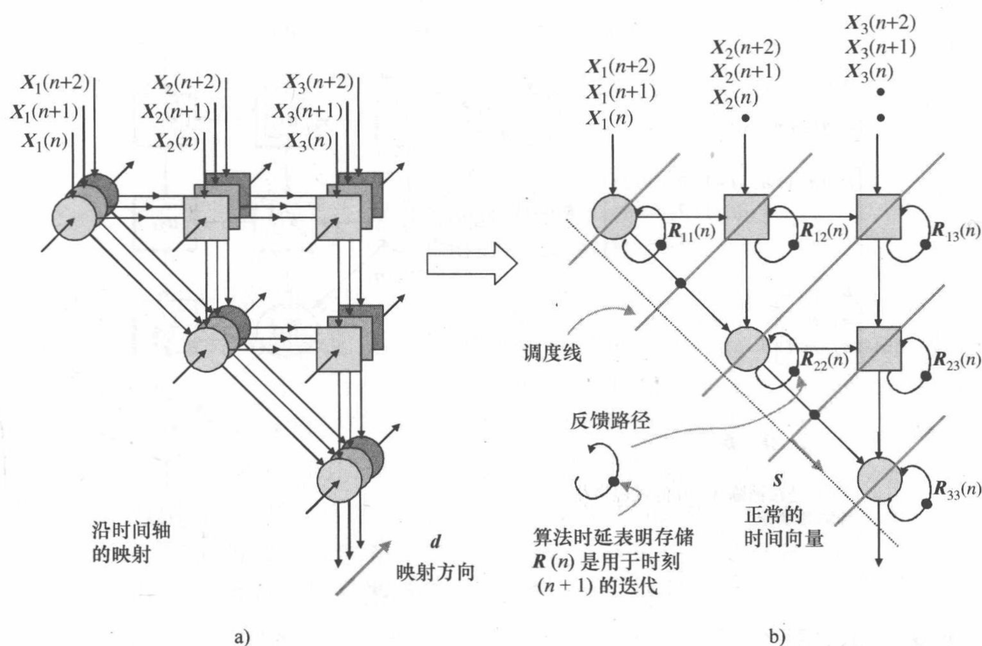


图 12-10　从 DG 到 SFG
a）DG　b）SFG

现的详细说明。特别强调的是要创造一个直观的可参数化的设计，因此未来实现的快速发展成为可能。

12.5.3　Givens 旋转的脉动实现

图 12-11 所示为由此产生的脉动阵列的常规 Givens RLS 算法。注意到在图 12-7 中，由 Gentleman 和 Kung（1981）提议的原始版本不包括沿着 BC 的对角线形成的余弦结构，并且这个版本将沿着对角线连接 BC 用箭头表示出来了。由 McWhirter（1983）做出了重要修正，因为它允许了 QR 阵列执行两个功能来计算权重，并且还可以使它能像滤波器本身一样运行，换言之，误差残留（后验误差）会在不需要权重向量提取的条件下被找到。比起自适应波束形成，此应用中提供了一个很有吸引力的解决方案。图 12-12 和图 12-13 所示分别为 BC 和 IC 的定义。

数据向量$\underline{x}^{T}(n)$是来自阵列顶部的输入，并且轮流地存储在三角形矩阵$\boldsymbol{R}(n-1)$的每一行中旋转被渐渐消除。旋转参数 c 和 s 在一个 BC 中得到计算，这样来消除输入 $x_{i,i}$（n）。然后这些参数不变地沿着 IC 传输，继续旋转。IC 的输出值 x_{i+j}，$j(n)$ 成为下一行的输入值。同时，新的输入被反馈进阵列的顶端，

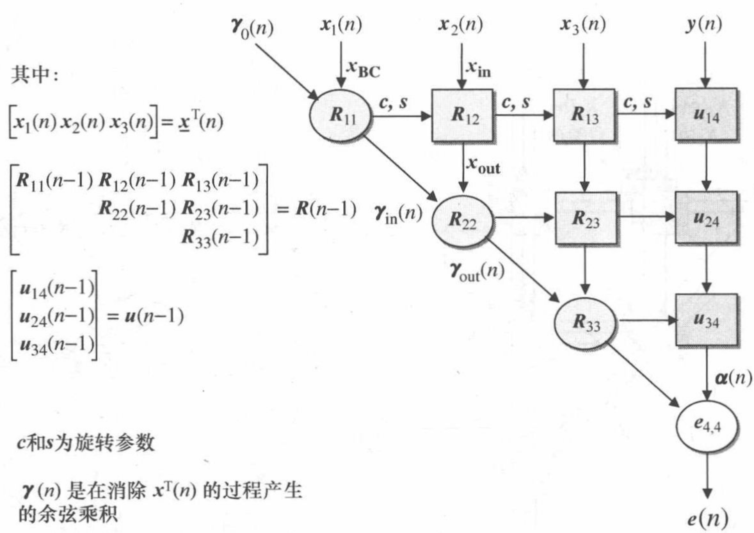


图 12-11 RLS 算法的脉动 QR 阵列

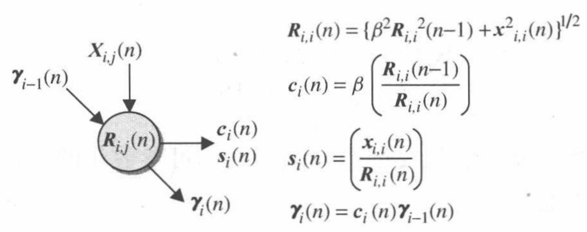


图 12-12 QR-RLS 算法的 BC

然后此过程不断重复。在这个过程中，$\boldsymbol{R}(n)$和$\boldsymbol{u}(n)$的值被更新然后存储在阵列中，在下一周期来临时使用。

对于 RLS 算法，在等式中需要包含遗忘因子 λ 和余弦的结果 $\boldsymbol{\gamma}$。因此，BC 和 IC 的运算已经相应地得到了修改。在阵列中一个记号已经被分配给了变量。每个 $\boldsymbol{R}$ 和 $\boldsymbol{u}$ 都有一个脚注，被标为（i，j），它们代表 $\boldsymbol{R}$ 矩阵和 $\boldsymbol{u}$ 向量中元素的位置。一个相似的记号被分配给了 $\boldsymbol{X}$ 输入和输出的变量。图 12-12 和图 12-13 所示分别为对更新的 BC 和 IC 单元的描述。脚注是关于 QR 阵列中单元位置的坐标。

12.5.4 二次方的 Givens 旋转

在 BC 单元中对标准 Givens 旋转有除法运算和二次方根运算两种方法，如图

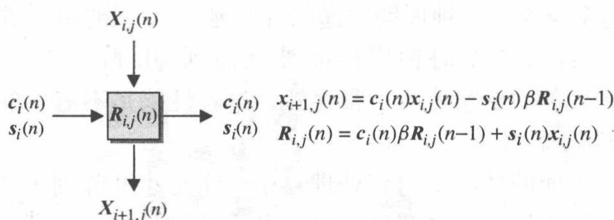


图 12-13　QR-RLS 算法的 IC

12-12 所示。对推导 Givens 旋转 QR 算法有着很广泛的研究，这是为了避免复杂的运算，同时减少计算的总量（Cioffi 和 Kailath 1984，Döhler 1991，Hsieh 等. 1993，Walke 1997）。一个可行的 QR 算法是 Givens 旋转的二次方（SGR（Döhler 1991））。在这里，Givens 算法可以避免对 BC 中二次方根运算及 IC 中半数乘法器的需要。Walke（1997）的研究指出这个算法在自适应波束形成中以合理的字长提供了出色的性能（甚至连同小数部分一起字长才 12 位，比起递归环中的字长要长了 4 位）。证明这个算法是一个自适应波束形成设计的合理选择。图 12-14 所示为 SGR 算法中的 SFG，并且包括了对 BC 和 IC 的描述。

因为减小的字长和降低的运算要求，这个算法仍然要求浮点算法的动态范围，但是提供了定点算法来减小体积。它有允许一个乘加器来更新 R 的额外优势。递归环的关键优势是简易，因为这个环中的时钟周期数将控制一个特殊时钟频率的最大排出量。例如，如果一个 QR 阵列在递归环中有 10 个时钟周期，则连续的 QR 迭代之间就会需要 10 个时钟周期，由此使得 $\boldsymbol{R}(n)$能在 $n+1$ 迭代中使用、计算。

当硬件中的消耗很少、允许减小全部的字长时，能够通过增加累加器 R 的字长来提高精确度。这被称为增强的 SGR（Enhanced SGR，E-SGR）算法（Walke 1997）。

然而，即使通过 SGR 算法降低实现的水平，QR 单元的复杂度仍然很大。此外 QR 阵列中处理器的数量连同输入的数量呈二次方增加，这样对一个 N 输入的系统而言，需要$(N^2+N)/2$ 个 QR 处理器。此外，为每个单元实现处理器中的数据速率会比那些许多应用要求的数据速率快很多。接下来的小节将详述为实现 SGR QR－RLS 算法，推导出一个高效通用架构的过程。

12.6　高效结构设计

随着 SGR QR－RLS 算法的复杂度和输入处理器数量的大幅度提高，特定的生成一个高效的、满足性能要求的、低开销的 QR 阵列架构是很重要的。这样一个应用包含有 40 个输入，但是只需 1MSPS 的吞吐量。实现全 QR 阵列，使用

200MHz 的时钟速率及 4 个时钟周期的递归环时延（一个被用于贯穿本章的值）。这意味着新的输入经每 4 个时钟周期能被反馈到 QR 阵列中，因此可以提供 200MHz/4 = 50MSPS 的吞吐量能力。很显然，这种性能是不需要的，并且这个设计可能受益于硬件的缩减。

这是通过将三方面的功能映射到处理器中一个更小的阵列来实现的。除了沿着对角线的 BC 运算的位置外，QR 阵列的三角形使推导出一个高效架构的过程复杂化了。图 12-15 所示为一个简单的 QR 单元映射到一个线性架构的例子，从左到右映射到 N 个处理器上。这个映射中存在两个问题：第一，BC 和 IC 运算是映射到同一个处理器上的，从图 12-14 中可以看出这些运算之间有明显的区别；第二，被映射架构的处理器不能被有效使用，只有第一个被全部利用。此效率降低了处理器的容量，导致总效率只能在 60% 左右，这样的资源利用率不是很乐观。

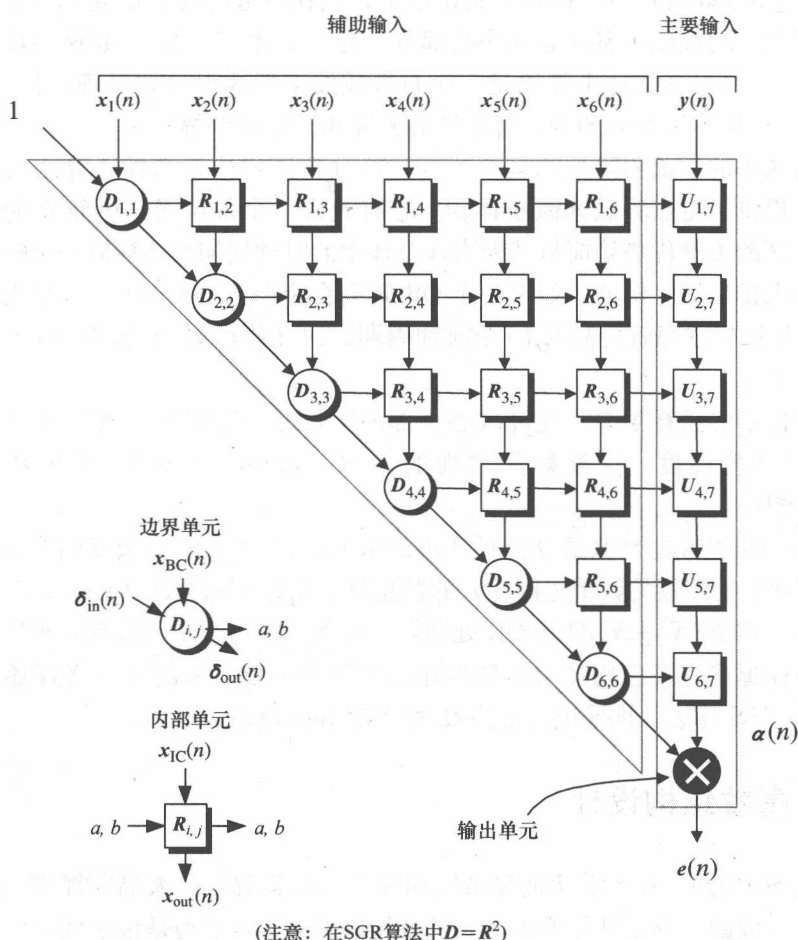


图 12-14 二次方的 Givens 旋转 QR-RLS 算法

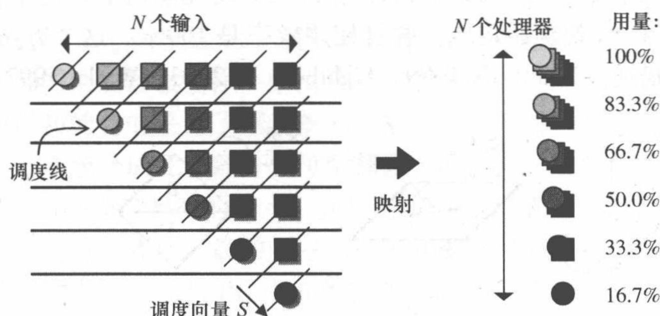


图 12-15　简单的线性阵列映射

Rader（1992，1996）通过在分配运算给处理器之前的三角形操作中生成一个高效架构，如图 12-16 所示。QR 阵列的 B 部分在 x 轴被折叠并且反折到剩下的阵列上。这就导致单元中有一个矩形阵列，它能被映射到一个由 $N/2$ 个处理器组成的线性架构上。但是，这些处理器仍需要执行 QR 运算。一个选择是在有很强架构相似度的 BC 和 IC 之间使用 CORDIC 算法（Hamill 1995），另一个选择是设计一个基于核心算法模块的通用 QR 单元（Lightbody 等　2007），在此模块上可建造单元功能性。QR 阵列的其他映射也是存在的，其中一个（Tamer 和 Ozkurt 2007）是使用一个瓦片结构来在上面映射 QR 单元，这就导致处理器只能执行 IC 运算并且也只能执行这两种功能。

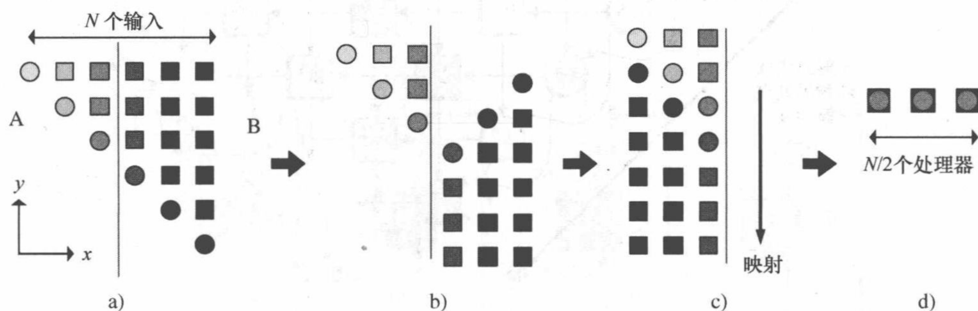


图 12-16　Radar 映射（Radar 1992，1996）

另一个映射（Walke 1997）是设法维持一个有效率的架构同时维持 BC 和 IC 运算来区别处理器。这是通过巧妙地操纵三角形阵列的形状，以便将所有的 BC 操作对齐到矩形阵列处理器的一列上，同时所有的 IC 运算被映射到其他行上。这是通过折叠和旋转 QR 阵列的一部分来做到的，如图 12-17 所示，是一个 7 输入三角形阵列的例子。结果就是将 $2m^2+3m+1$ 个有着本地互连性单元（也就是

$N=2m+1$ 个输入）的三角形矩阵映射到一个线性架构上，这个架构由一个 BC 处理器和 m 个 IC 处理器构成，并且使用效率是 100%。这个方法在别的地方有更加详细的描述（Lightbody 1999，Lightbody 等 2003，Walke 1997）。

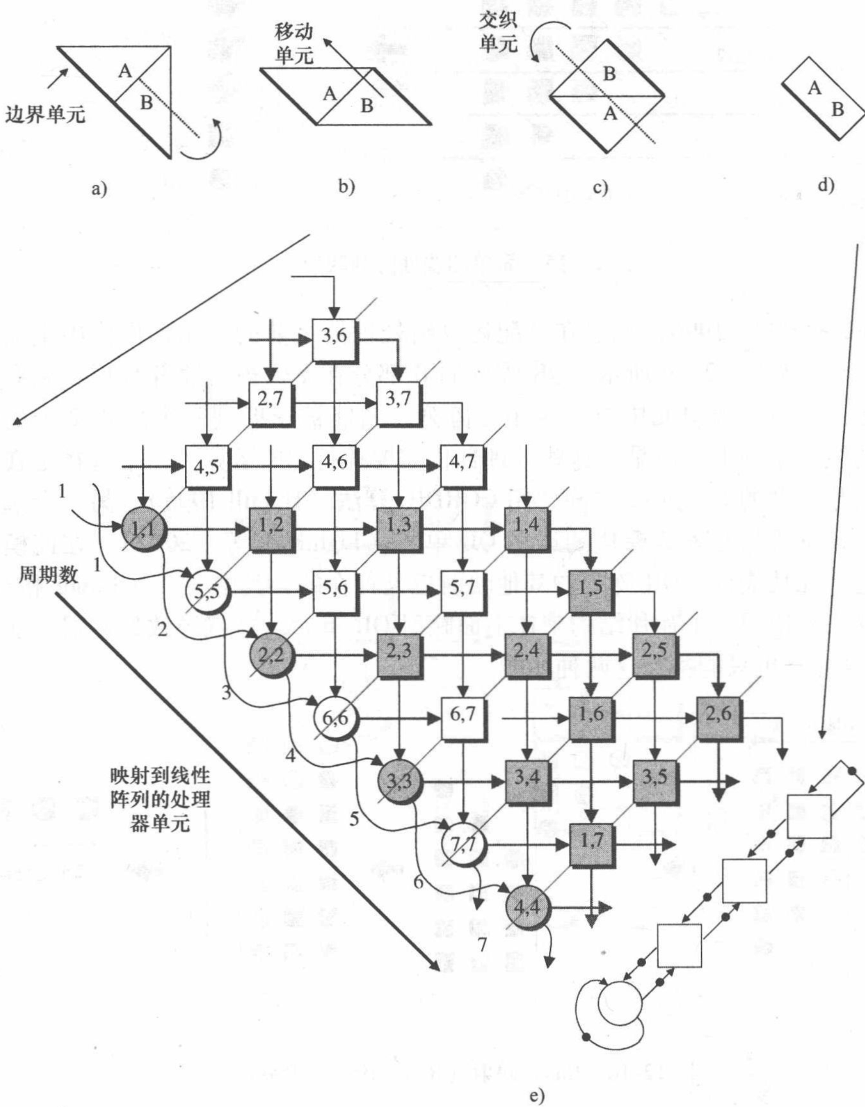


图 12-17 交错处理器阵列。

摘自 *Diesign of a Parameterizable Silicon Intellectual Property Core for QR-Based RLS Filtering*, by G. Lightbody & R. Woods，IEEE Trans on VLSI Systems，Vol. 11，No. 4，© 2003 IEE

a）三角形阵列 b）修改后的数组 c）矩形阵列 d）本地连接阵列

e）映射到一个线性阵列本地连接的处理器单元

为了标注清楚，每个 QR 运算被分配了一个由这个运算计算的 R（或者 U）的坐标，比如运算 $R_{1,2}$ 由坐标 1,2 表示，运算 $U_{1,7}$ 由坐标 1,7 表示。为了简化解释，阵列底部的乘法器被当作一个由 7,7 表示的 BC。

这个映射最初的目的是为了调动单元，来让它们形成一个本地互连的有规律的矩形阵列。这能被分成几个部分，每个部分被分配到一个单独的处理器上。以这种方法可以实现单元 100% 的使用，并且得到一个邻域上最邻近的阵列。可以通过以下 4 步得到矩形阵列。最初的三角形矩阵被分成两个更小的三角形矩阵 A 和 B。在第（$m+1$）个 BC 与 BC 成直角后进行切割。三角形矩阵 A 移位到一个 $m+1$ 行和 $m+1$ 列的三角形矩阵列的底部。

现在需要移动三角形矩阵 B 来使它移动到矩形阵列的顶部。这需要分两步。首先映射 B 到 x 轴上，使 B 与 BC 排列起来并以这种方式来使得它们在三角形矩阵 A 中平行于 BC，形成一个平行四边形，如图 12-17b 所示。然后被映射的三角形矩阵 B 沿着 y 轴被提升并且沿着 x 轴向左移动到矩形阵列 A 的上方，形成矩形阵列（见图 12-17c）。正如描述的那样，BC 运算被排列成两行，因此对于分配运算到一个线性架构上而言，矩形阵列仍然不是一个合适的形式。

下一步目的是将大的矩形阵列对折来使得 BC 运算的两列沿着一列排列。单元的折叠交错可以得到一个紧凑的矩形处理器阵列（见图 12-17d）。从这个矩形的处理器阵列中，通过映射对角线上的单元到一个线性阵列上，可将所有 BC 运算分配到一个 BC 进程上，所有的 IC 运算分配到 m 个 IC 处理器上（见图 12-17e）的简化架构。图 12-18 所示为更加详细的最终的线性架构。

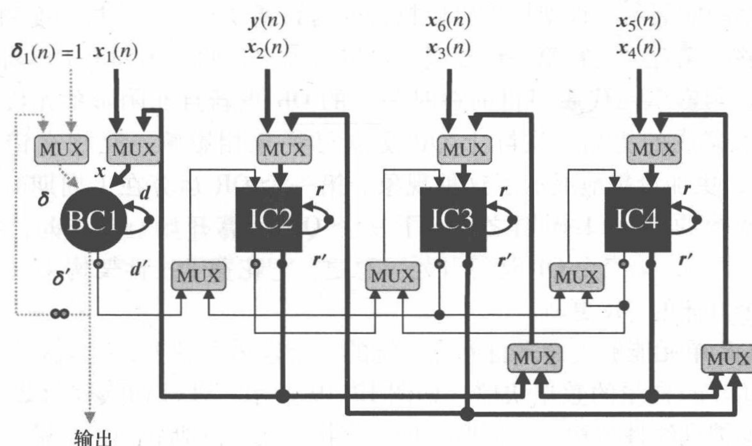


图 12-18　一个 7 输入 QR 阵列的线性架构。摘自 *Design of a Parameterizable Silicon Intellectual Property Core for QR-Based RLS Filtering*，by G. Lightbody & R. Woods，IEEE Trans on VLSI Systems，Vol. 11，No. 4，© 2003 IEEE

图 12-17e 所示运算矩阵列中处理器的每一行都有线穿插在其中。这些线代表了那些在最后的线性阵列架构的每个周期需要执行的运算。这就是所谓的运算调度，并且通过调度向量 s 被更加紧凑地表示，调度向量 s 是一个与调度线垂直的箭头。在分析的这一阶段，假设每个单元处理器占一个时钟周期。最终，线性阵列的所有处理器单元输出上都有寄存器来维持这个调度。数据选择器被放置在 QR 单元的输入上来控制数据输入，不管是来自系统的输入还是来自邻近单元的输入。底部的数据选择器是管理在原始阵列的行之间的数据流方向。

原始的 QR 阵列单元存储的 R 值从一个迭代到下一个。这样的存储为了可以运行简化的架构，因此要求在多重时钟周期的单元递归环中存储大量的 R 值。有一个方法是保持这个值在本地 QR 单元的循环数据通道中，而不是在外部存储器中，也就是这些值在本地被流水线化来对其进行延时，直到它们被需要时就不再延时。递归环中一些要求的时延可以通过环中现存的运算时延得到满足，并且剩余的时延也可以通过插入额外的寄存器来得到实现。一些规格的外部存储器是合适插入的。

12.6.1 调度 QR 运算

架构的派生仅仅是必要发展的一部分。更复杂的工作是确定一个有效的计划，以确保每组操作所需数据被执行的时间，同时保持一定的效率。这暗示了数据一定是以调度向量的方向跨调度线流动的。图 12-17e 中的矩形处理器阵列包含了 QR 算法要求的所有运算，展示了一系列能在线性架构上实施的情况。因此，用这个图能展示线性架构上可以执行的运算调度。一个分析调度和定时的问题现在能够被完善。观察第一调度线，可以看出来自两个不同 QR 更新的运算已经被交错。阴影单元代表了目前在时刻 n 的 QR 更新且非阴影单元代表了先前 $n-1$时刻未完成的更新。实际上 QR 更新已经互相影响、互相交错了。如图 12-19所示，更加清楚地展示了这种现象。第一个 QR 运算在 1 周期时开始，然后在线性架构的 $2m+1$ 周期之后，下一个 QR 运算开始。同样的，在下一个 $2m+1$周期之后，第三个 QR 运算开始。总之，它花费了线性架构 $4m+1$ 个周期来完成一个具体的 QR 更新。

需要 QR 单元能够支持来自外部系统的 x 个输入的快照，也就是形成输入 $\boldsymbol{x}(n)$矩阵和 $\boldsymbol{y}(n)$向量的数据快照，如图 12-19 所示。外部输入经每 $2m+1$ 个时钟周期被反馈到线性架构中。同时它们也会接受线性阵列式内部的输入。映射过程已经能使这些互连被保持在本地，这是一个主要的优点。

如果每一个 QR 单元占用一个单个时钟周期来产生一个输出，那这将不会有如图 12-17 所示的调度妨碍。但是，一定要考量额外的定时问题，因为每个 QR 单元中的进程单元都有详细的定时要求。在第 12.8 节中会详细讨论运算的重

定时。

注意到图 12-19 中突出的处理器阵列式，等同于图 12-7e 中给出的处理器阵列。该处理器阵列是一个关键起点，从这个起点可以发展一个通用 QR 架构。

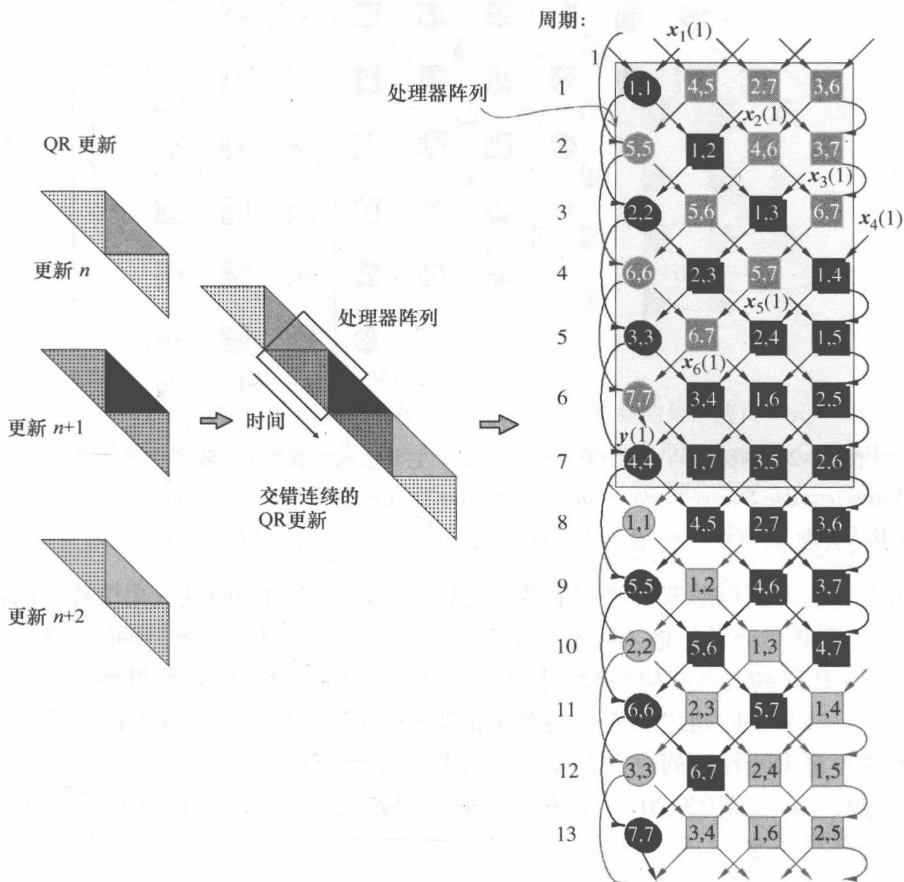


图 12-19　交错的连续 QR 运算。摘自 *Design of a Parameterizable Silicon Intellectual Property Core for QR-Based RLS Filtering*, by G. Lightbody & R. Woods, IEEE Trans on VLSI Systems, Vol. 11, No. 4, © 2003 IEEE

12.7　通用 QR 单元

目前提出的技术已经被应用到仅有一个主要输入的 QR 阵列中，也就是一个 y 输入。为了发展一个通用QR架构，主要输入的数量需要是可变的。这会产生一个由一个三角形部分和一个矩形部分组成的 QR 阵列，如图 12-20 所示，它的大小由附属输入的数量和主要输入的数量决定。一般而言，输入到三角形部分的

图 12-20　有着 $2m+1$ 个附属输入和 p 个主要输入的通用 QR 阵列。摘自 *Design of a Parameterizable Silicon Intellectural Property Core for QR-Based RLS Filtering*, by G. Lightbody & R. Woods, IEEE Trans on VLSI Systems, Vol. 11, No. 4, © 2003 IEEE

数量要比输入到矩形部分的数量多，比如雷达应用的例子中到三角形部分的输入是 40 而到矩形部分的输入只有 2 个。已经提出了由三角形部分和矩形部分组成的通用 QR 阵列实现架构映射的技术好。不同层次的硬件映射都得到了应用，这提供了一些基于原始线性阵列的合适的架构。IC 处理器的数量也许会进一步减少，或者多重线性阵列也许会被结合起来，这都取决于应用的性能要求。注意到这些连接已经从图 12-20 和接下来的图表中移除了，这都是为了减少图表的复杂度和提升清晰度。

12.7.1　处理器阵列

在先前的章节中，QR 阵列的三角形结构转变成了本地互连的处理器的一个矩形处理器阵列，如图 12-17d 所示。从这个起点开始，运算能被映射到一个缩减的架构上。接下来的例子将证明一个简化创建处理器阵列的方法。

处理器阵列是通过两步获得的。首先如图 12-20 所示，通过将阵列的一角从右手边折叠到第 m 个单元之后。这个从矩阵的一个角被折叠的单元被交错到未被折叠的行之间，如图 12-21 所示。接下来，需要考虑连续的 QR 更新。图 12-21所示结构中的间隔通过当前 QR、先前迭代、下个迭代的更新，3 个 QR 的交错来被移除。这在线性阵列的派生过程中同样有效，图 12-19 所示为没有矩形部分的原始三角形阵列。

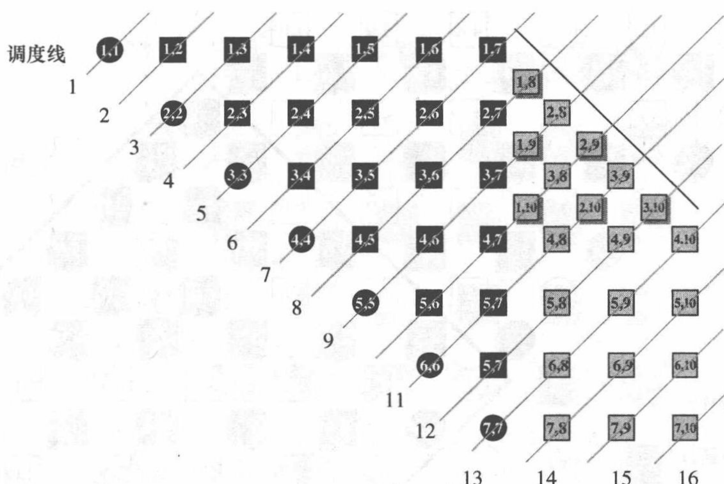


图 12-21　一般的 QR 阵列：从右手边第 m 个单元折叠的角。摘自 *Design of a Parameterizable Silicon Intellectural Property Core for QR-Based RLS Filtering*, by G. Lightbody & R. Woods, IEEE Trans on VLSI Systems, Vol. 11, No. 4, © 2003 IEEE

折叠位置的选择及阵列三角形部分的大小是很重要的。通过将折叠部分放置在右手边的第 m 个单元之后，能得到一个有规律运算的矩形阵列。这能够更详细地展示出来，比如在图 12-22 中的一般 QR 阵列。正如三角形阵列一样，应用于三角形部分和矩形部分的同样过程产生了一个随着时间重复的部分，并且包含了所有被要求的 QR 运算。这个部分指的是处理器阵列。如图 12-23 所示，更加清楚地描述了处理器阵列，其中展示了来自图 12-22 中重复的部分。

在图 12-23 所示例子中，处理器阵列包含了由 3 个连续 QR 更新的 QR 运算，由不同的阴影单元代表。互连已经被维持在这个图形中，它突出了单元的本地互连性。处理器阵列的大小由三角形 QR 阵列的原始大小决定，也就是说，附属输入和主要输入的数量分别是 $(2m+1)$ 和 p 个。最后的处理器阵列有 $(2m+1)$ 行和 $(m+p+1)$ 列，结果就是在原始阵列中给予了单元总的数量。在这个处理器阵列中，有着一些硬件缩减级别可变的架构，能通过将阵列分成几个部分并且将每个部分分配到一个单独的处理器上来获得。以下有 QR 架构的几个可能的变式：

线性架构：矩形阵列被映射到一个有单个 BC 和 $(m+p)$ 个 IC 的线性阵列上。

矩形架构：矩形阵列被映射到单元的大量线性行上。这个架构会有 r 行（其中 $1<r=2m+1$），并且每行会有一个 BC 和 $(m+p)$ 个 IC。

稀疏线性架构：矩形阵列被映射到一个有单个 BC 和至多 $(m+p)$ 个 IC 的

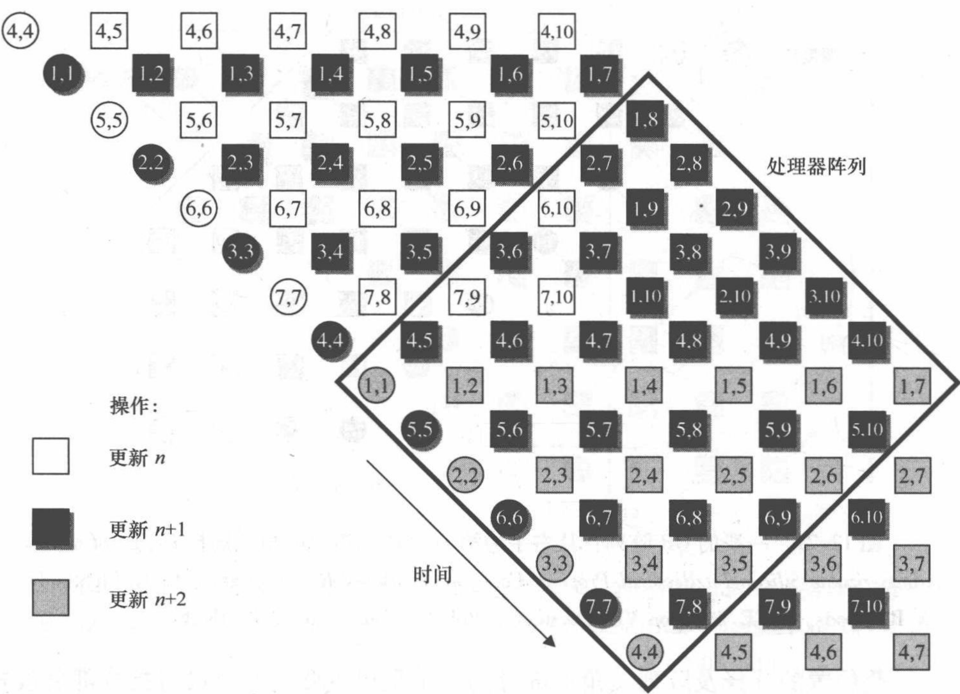


图 12-22 重复部分。摘自 *Design of a Parameterizable Silicon Intellectual Property Core for QR-Based RLS Filtering*, by G. Lightbody & R. Woods, IEEE Trans on VLSI Systems, Vol. 11, No. 4, © 2003 IEEE

线性架构上。

稀疏矩形架构：矩形阵列被映射到单元的许多线性行上。这个架构会有 r 行，（其中 $1<r=2m+1$），并且每行会有一个 BC 和至多（$m+p$）个 IC。

以上摘自 Lightbody *et al.*，©2003 IEEE。

通过使用与图 12-23 中相同的 QR 处理器阵列例子，在下面的部分给出了每种类型的简化架构的例子。

1. 线性阵列

线性阵列是源于将处理器阵列运算的每列分配到一个单独的处理器上，这就生成了 $m+p+1$ 个处理器的一个线性架构，如图 12-24 所示。总之它花费了线性阵列的 16（也就是 $4m+p+1$）个周期来完成每个 QR 运算。此外，在连续的 QR 更新开始之前也要 7（也就是 $2m+1$）个周期。这个值被记作 T_{QR}。注意到目前为止，QR 单元的时延是一个时钟周期，也就是在每个时钟周期上，QR 运算的每行都在线性架构上被执行。同样，假设递归环仅仅有 1 个周期的时延。之后的章节会检测多重周期时延的影响，当有着详细定时的单元处理元件被用于一般 QR 架构的发展时，它才会发生。

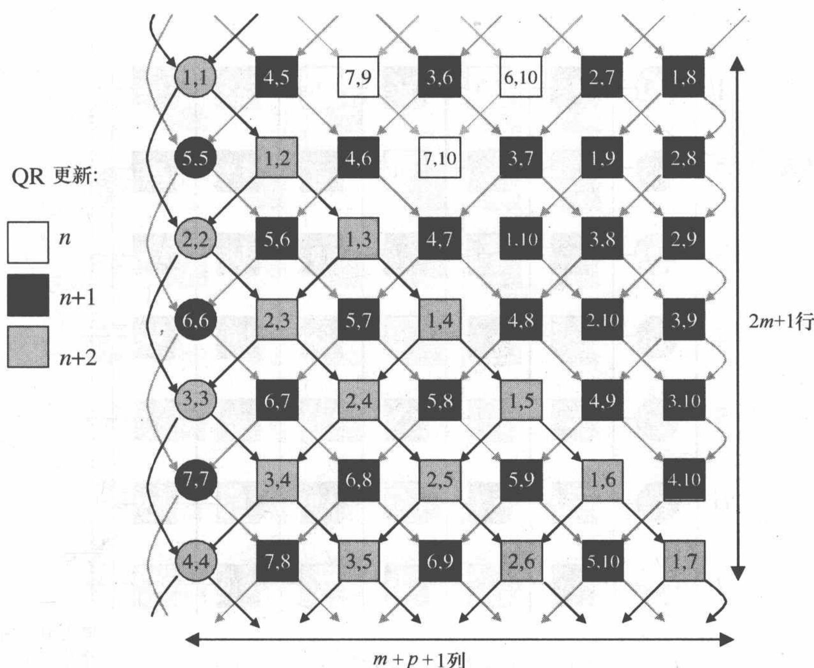


图 12-23　处理器阵列。摘自 *Design of a Parameterizable Silicon Intellectural Property Core for QR-Based RLS Filtering*, by G. Lightbody & R. Woods, IEEE Trans on VLSI Systems, Vol. 11, No. 4, © 2003 IEEE

2. 稀疏线性阵列

图 12-25 所示为进一步的硬件缩减，这就产生了一个稀疏线性阵列。在这里 IC 处理器的数量已经减半。当 IC 运算的多重列（也就是列）被分配到每个处理器时，架构的迭代数量因为这个因素被增加。因此，对于稀疏的线性阵列，T_{QR} 表示为 $2m+1$（被用于线性阵列）和 N_{IC}。图 12-26 所示为对稀疏线性阵列调度的例子。

3. 矩形阵列

处理器阵列能被行分而不是列分，使得 QR 运算的大量行被分配到处理器的一个线性阵列上。图 12-27 所示为被映射到一个阵列架构的处理器阵列上。因为处理器阵列由 7 行组成，其中 4 行被分配到一条线上并且另外 3 行被分配给另一条线。要平衡每一个线性数组的行数，就需要一个虚拟行的操作，并且通过被字母 D 标记的单元表示。

在每个时钟周期上，矩形阵列处理器执行原始处理器阵列的两行。每个 QR 迭代要花费 18 个周期来完成，这比线性阵列多花费了 2 个时钟周期，这些都是由于运算的虚拟行多而导致的。但是，QR 更新起动更加频繁了。这样的话，相比于花费 7 个周期的线性阵列，T_{QR}就是 4 对阵列架构，T_{QR}由下式决定：

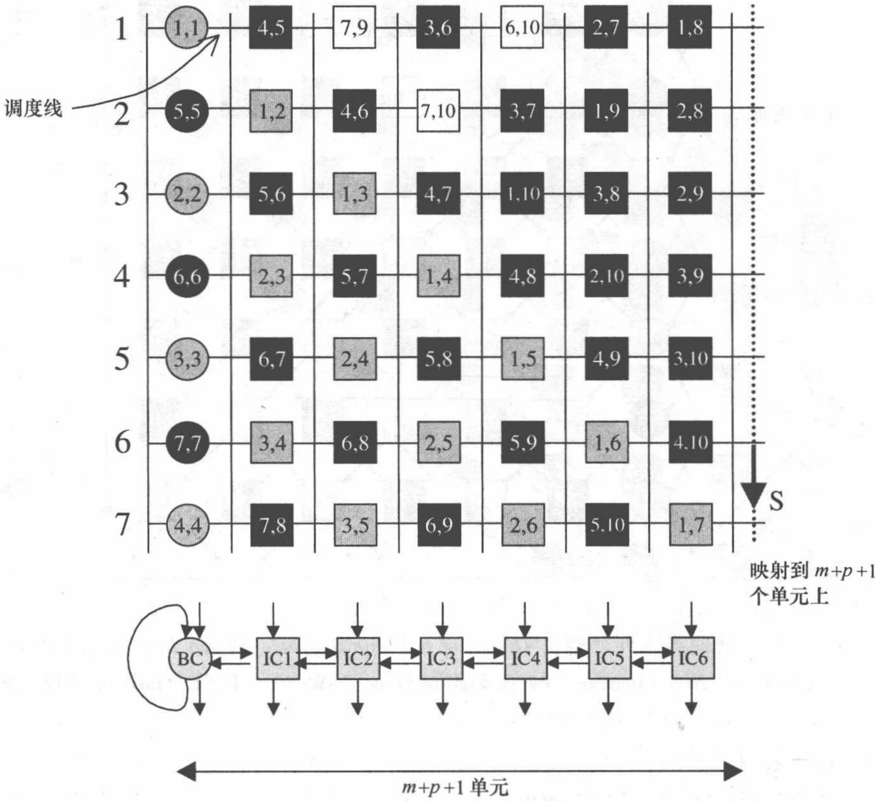


图 12-24 线性阵列。摘自 *Design of a Parameterizable Silicon Intellectual Property Core for QR-Based RLS Filtering*, by G. Lightbody & R. Woods, IEEE Trans on VLSI Systems, Vol. 11, No. 4, © 2003 IEEE

$$T_{QR}=\frac{(2m+1)+N_D}{N_{rows}}$$

式中，N_{rows}是矩形架构中处理器的线的数量，N_D 是需要用于平衡调度的虚拟运算的行的数量。结果值与被要求执行处理器阵列中所有运算的架构的周期数相关。

4. 稀疏矩形阵列

稀疏矩形阵列运算分配到稀疏线性阵列的多重行上。处理器阵列的许多行被分配到每个线性阵列上。列也被分区来使得运算的多重列被分配给每个 IC 处理器，如图 12-28 所示。

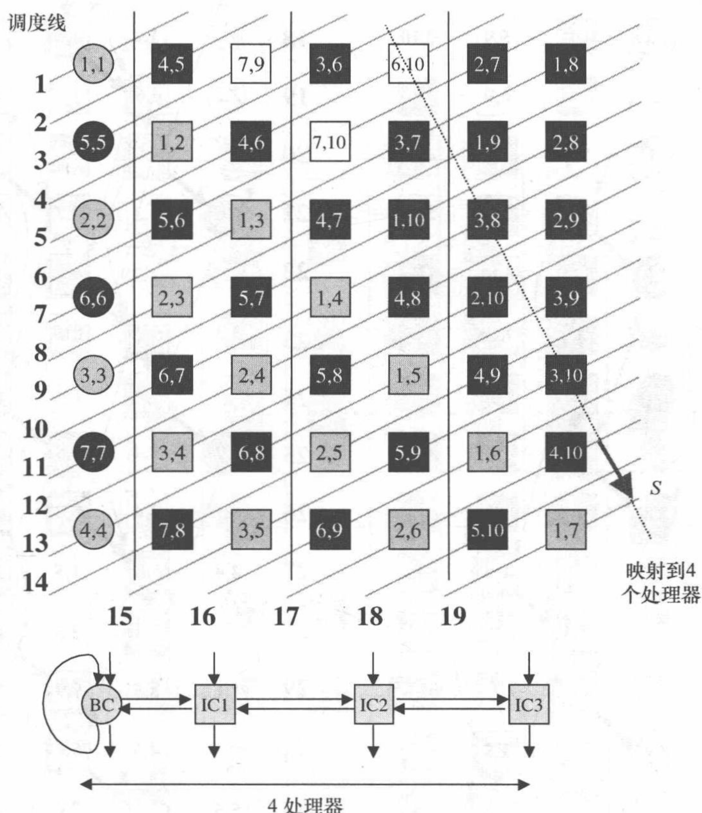


图 12-25 稀疏线性阵列。摘自 *Design of a Parameterizable Silicon Intellectual Property Core for QR-Based RLS Filtering*, by G. Lightbody & R. Woods, IEEE Trans on VLSI Systems, Vol. 11, No. 4, © 2003 IEEE

QR 更新花费 34 个周期来完成并且每 7 个周期起动一个更新，也就是 $T_{QR}=7$。包括了 N_{IC}的等式 T_{QR}变成了

$$T_{QR}=\frac{((2m+1)+N_D)N_{IC}}{N_{rows}}$$

例如，$T_{QR}=((2\times3+1+0)\times2)/2=7$ 个周期。

迄今为止的讨论仍集中在映射到有着奇数附属输入数的 QR 阵列上。这项技术可以被应用到一个偶数阵列上，但是总效率会有少量的降低。

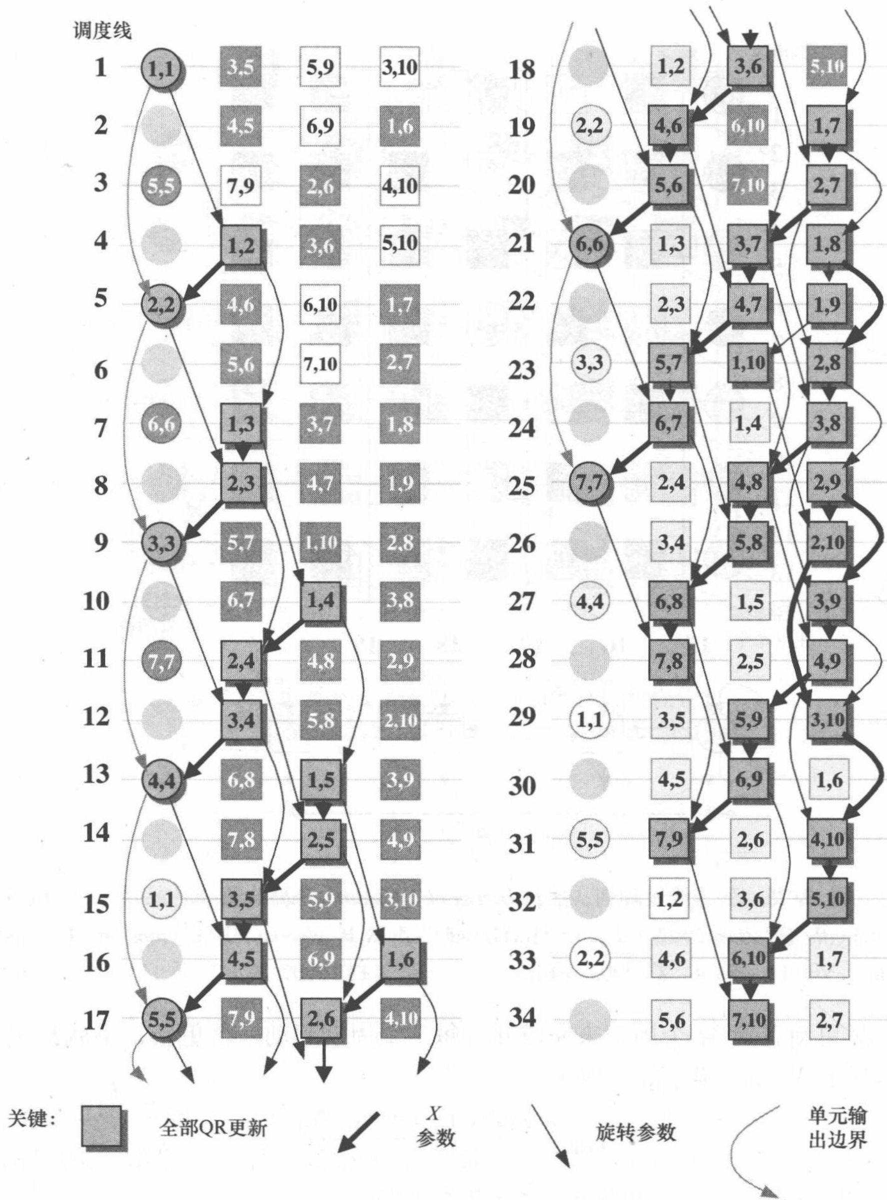
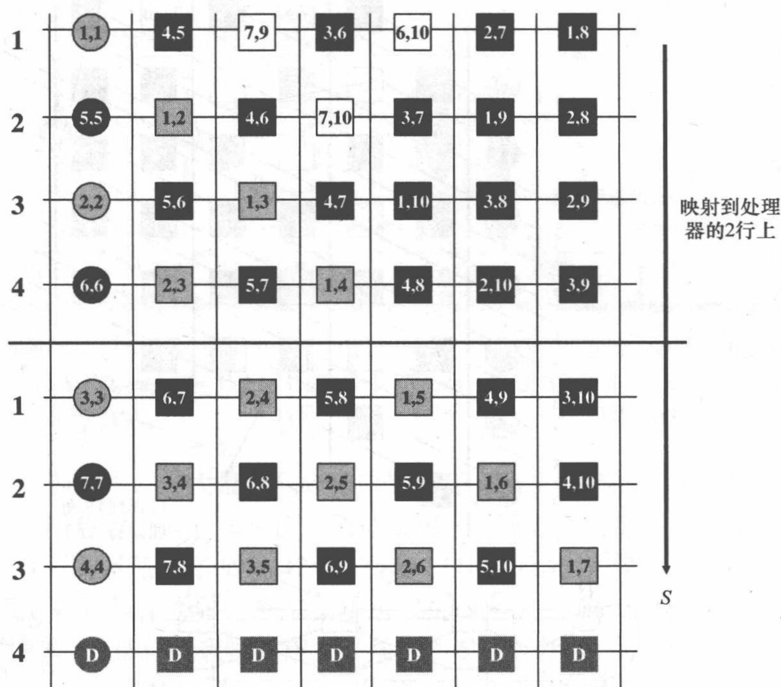


图 12-26 稀疏线性阵列上的一个 QR 更新调度


1
2
3
4
1
2
3
4
1,1
4,5
7,9
3,6
6,10
2,7
1,8
5,5
1,2
4,6
7,10
3,7
1,9
2,8
2,2
5,6
1,3
4,7
1,10
3,8
2,9
6,6
2,3
5,7
1,4
4,8
2,10
3,9
3,3
6,7
2,4
5,8
1,5
4,9
3,10
7,7
3,4
6,8
2,5
5,9
1,6
4,10
4,4
7,8
3,5
6,9
2,6
5,10
1,7
D
映射到处理器的2行上
S

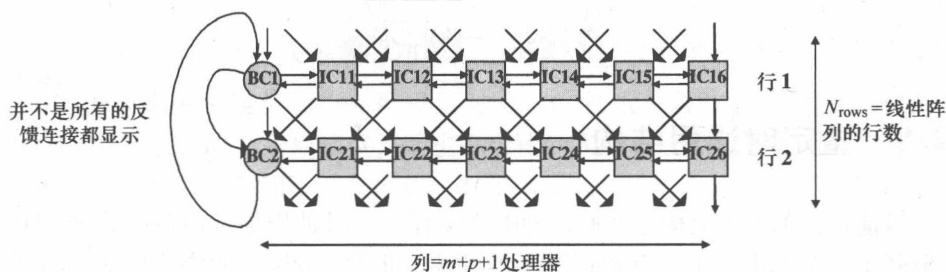

并不是所有的反馈连接都显示
BC1
IC11
IC12
IC13
IC14
IC15
IC16
BC2
IC21
IC22
IC23
IC24
IC25
IC26
行 1
行 2
N_{rows}=线性阵列的行数
列=$m+p+1$处理器

图 12-27　矩形阵列

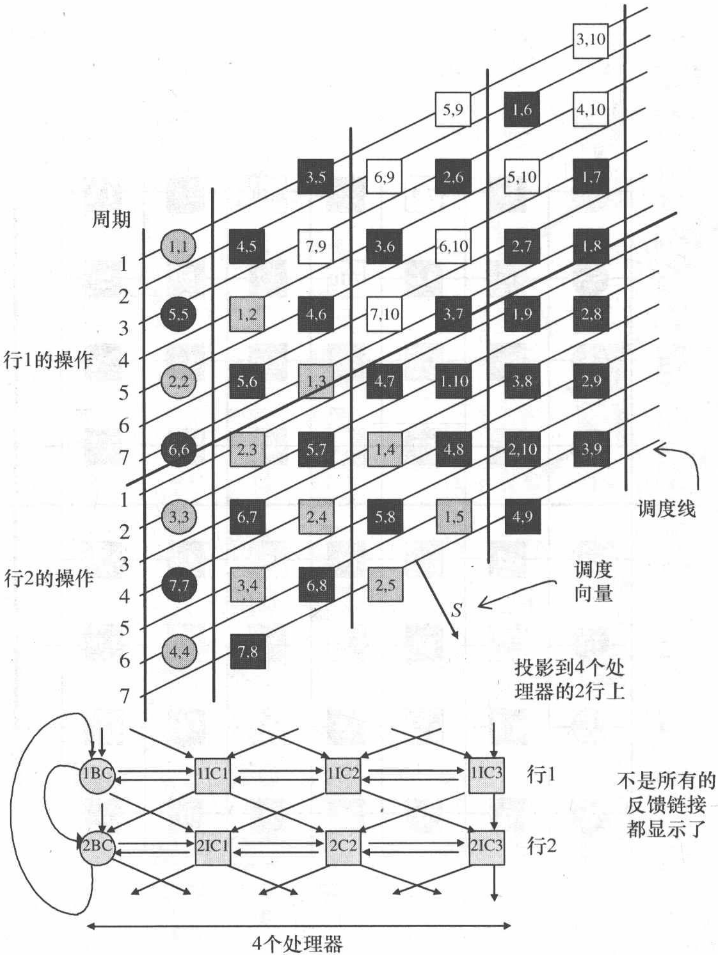


图 12-28 稀疏矩形阵列

12.8 重定时通用结构

目前讨论的 QR 架构已经假设 QR 单元有一个时钟周期的时延。由于架构的映射基于这个原因，不会有数据输入的冲突。但是，QR 单元中实际定时细节的内容将影响有效数据调度的保证。算法的 IP 处理器（McCanny 等 1997，University N 2007）被用于实现关键算法的功能，比如乘法、加法和除法，包括会影响整个电路定时的定时细节。这曾在第 8 章中有详细讨论。重定时的负面影响会导致 QR 单元的输出数据通道中的可变时延。本节探讨了在实时 QR 单元内使用真正硬件的结果。QR 阵列的选择是为了使用浮点算法来支持算法中变量的

动态范围。浮点库被用于支持变量的字长和流水线级，如图 12-29 所示。

浮点模块符号	Add	SubAdd	ShftSub	Mult	Div	Round
函数	加法: S=A+B	加法/减法: 当Sub=0时, S=A+B 其他情况S=A-B	移位减法: D = A - Shift (A,N).	乘法: P=X×Y	除法: Q = N/D	圆
符号	A, B, S	A, B, Sub, S	A, D	X, Y, P	N, D, Q	F_{in}, R, F_{out}
延时	0 - 3	0 - 3	0 - 1	0 - 2	0 - Mbit+1	0 - 1
延时的标号	P_A	P_A	P_S	P_M	P_D	P_R

图 12-29　算法的模型。摘自 *Design of a Parameterizable Silicon Intellectual Property Core for QR-Based RLS Filtering*, by G. Lightbody & R. Woods, IEEE Trans on VLSI Systems, Vol. 11, No. 4, © 2003 IEEE

在自适应波束形成中，像大量信号处理应用一样，表示复杂的算法是被需要的，同样，输入信号包含了大小和相位。通过使用一个信号作实部，另一个作虚部来使之得到实现，并且给出了如图 12-30 所示 SFG 展示的 BC 和 IC 运算。浮点的复杂乘法是由 4 个实乘法和 2 个实加法组成的，也就是$(a+jb)(c+jd)=(ac-bd)+j(ad+bc)$。最佳化通过使用 3 个乘法和 5 个加法/减法，可用于实现复杂的乘法。但是，考虑到因为指数运算代价较高，所以一个加法对浮点算法中的乘法而言是一种简单的计算，这并不是有益的。由于这个原因，4 次乘法的版本得到了使用。复杂算法运算的细节如图 12-31 所示。

在图 12-32 和图 12-33 中分别给出 BC 和 IC 的 SFG。这些图表展示了单元架构中算法模式的互连性。大多数的功能都是无需加以说明的，除了移位减法器。对很小的值 x 来说，运算 $\sqrt{1-x}$可以通过 $1-x^2$ 被近似算出，通过由 D = A - Shift（A，N）表示的一系列移位，这也许会得到实施。这个运算被用于实现 QR 单元的反馈路径中的遗忘因子 β。这个 β 值接近 1，因此为了功能应用，x 被设为（$1-\beta$）。

如图 12-32 和图 12-33 所示，在 QR 单元中有大量的反馈环。一个 RLS 的存储单元中的 R 值是可迭代的。这些循环将是一个基本的限制，以实现吞吐量的速率接近时钟速率，更重要的是，这可能会导致电路利用率的降低。换句话说，甚至当使用一个全 QR 阵列时，计算中的时延新 R 值也会限制吞吐量。

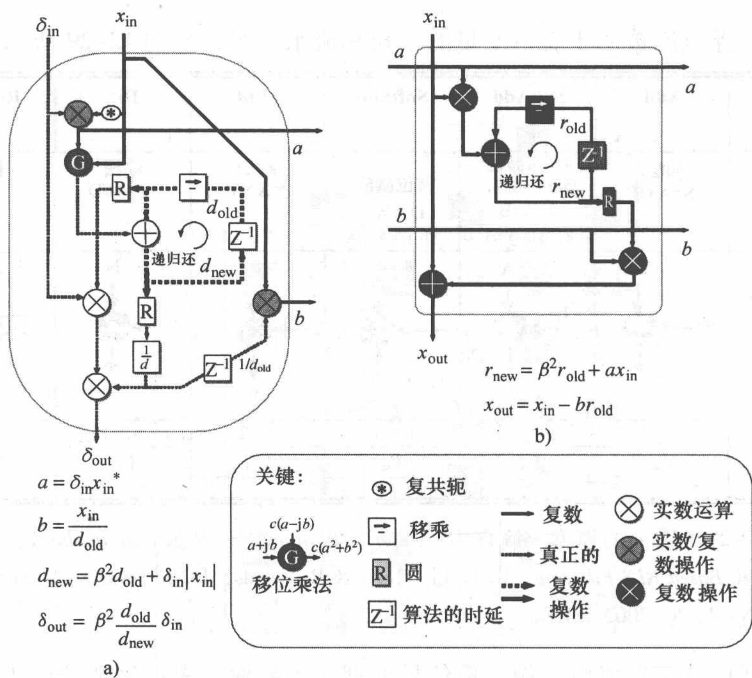


图 12-30　复杂算法的 SGR QR 算法的单元 SFG。摘自 *Design of a Parameterizable Silicon Intellectual Property Core for QR-Based RLS Filtering*, by G. Lightbody & R. Woods, IEEE Trans on VLSI Systems, Vol. 11, No. 4, © 2003 IEEE

a）边界单元　b）内部单元

功能	复数/实数乘法	复数加法	复数乘法	特殊功能
符号	$a+jb$；c；$ca+jcb$	$a+jb$；$c+jd$；$(a+c)+j(b+d)$	$a+jb$；$c+jd$；$(ac-bd)+j(ad+bc)$	$c(a-jb)$；$a+jb$；G；$c(a^2+b^2)$
实部件	a jb b；P_M P_M；ca cb	a c jb jd；P_A P_A；$a+c$ $j(b+d)$	a jb c jd；P_M P_M P_M P_M；P_A P_A；$ac-bd$ $j(ad+bc)$	a jb ca jcb；P_M P_M；P_A；$c(a^2+b^2)$
总延时	P_M	P_A	P_A+P_M	P_A+P_M

图 12-31　算法模型。摘自 *Design of a Parameterizable Silicon Intellectual Property Core for QR-Based RLS Filtering*, by G. Lightbody & R. Woods, IEEE Trans on VLSI Systems, Vol. 11, No. 4, © 2003 IEEE

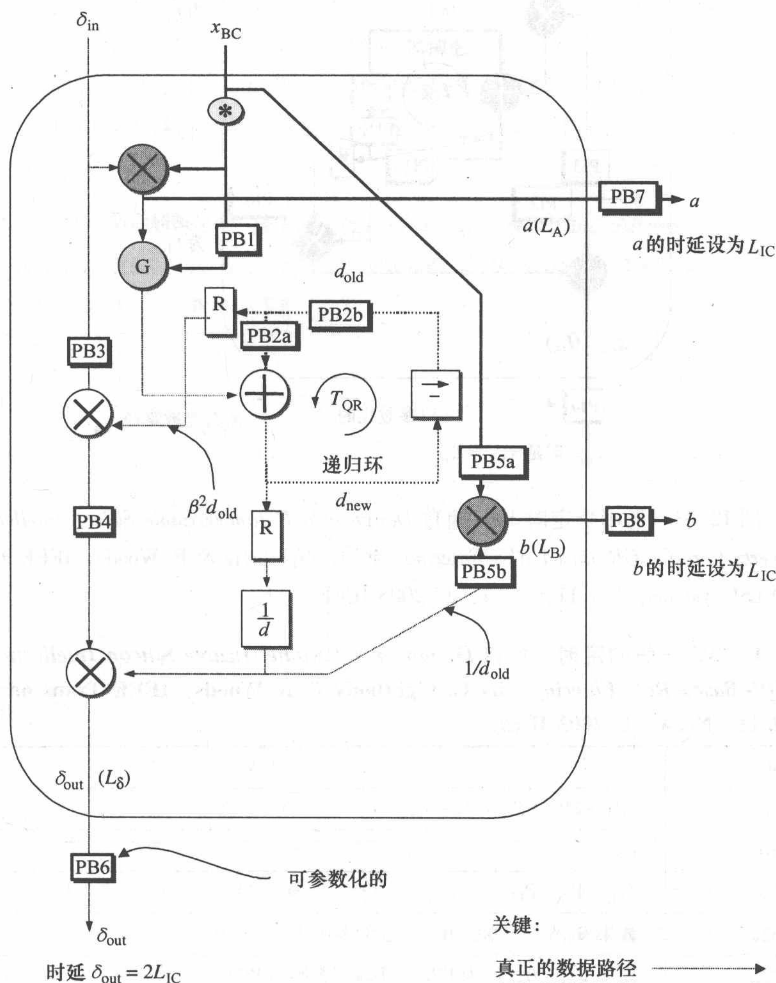


图 12-32 一般的重定时 BC。摘自 *Design of a Parameterizable Silicon Intellectual Property Core for QR-Based RLS Filtering*, by G. Lightbody & R. Woods, IEEE Trans on VLSI Systems, Vol. 11, No. 4, © 2003 IEEE

图 12-32 和图 12-33 描述了被放置在数据通道中的有着一般时延的 QR 单元。这些是为了允许运算的再同步，因为算子中可变的时延，也就是为了确保正确时间。在表 12-1 和表 12-2 中分别列出 BC 和 IC 的可编程时延的一般表达。

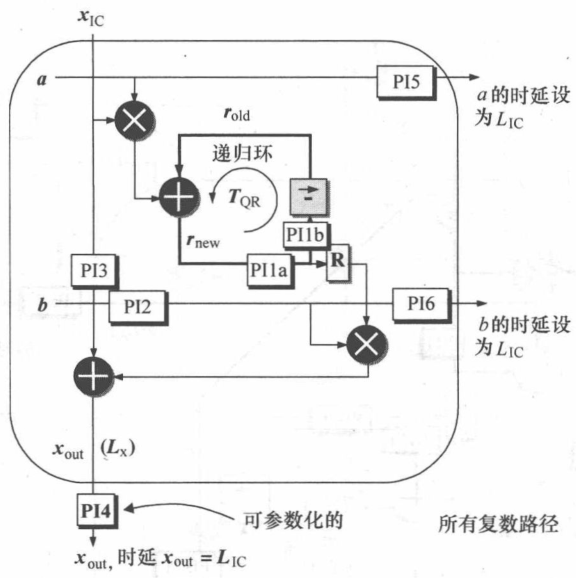


图 12-33 通用重定时 IC。摘自 *Design of a Parameterizable Silicon Intellectual Property Core for QR-Based RLS Filtering*, by G. Lightbody & R. Woods, IEEE Trans on VLSI Systems, Vol. 11, No. 4, © 2003 IEEE

表 12-1 BC 一般的定时。摘自 *Design of a Parameterizable Silicon Intellectual Property Core for QR-Based RLS Filtering*, by G. Lightbody & R. Woods, IEEE Trans on VLSI Systems, Vol. 11, No. 4, © 2003 IEEE

BC	时延值
B_{RL}	$2P_A + 2P_M + P_R + R_S - T_{QR}$
PB1	P_M
PB2	$T_{QR} - P_A - P_B$
PB2a	如果 $B_{RL} < 0$，则为 B_{RL}，否则为 0
PB2b	如果 $B_{RL} < 0$，则为 PB2 - PB2a，否则为 PB2
PB3	如果 $B_{RL} > 0$，则为 B_{RL}，否则为 0
PB4	$2P_A + P_M + P_R + P_D - PB3$
PB5	$2P_A + 2P_M + P_R + P_D - T_{QR}$
PB5a	如果 PB5 < 0，则为 PB5，否则为 0
PB5b	如果 PB5 > 0，则为 PB5，否则为 0
PB6	$L_{IC} - L_\delta$
PB7	$L_{IC} - L_a$
PB8	$L_{IC} - L_b$

还有为了维持一个有规律的数据调度，QR单元的时延得到了调整，使x的值和旋转参数同时来自于QR单元的输出。这些输出中的IC时延通过使用L_{IC}来表达。在产生旋转参数a和b中BC的时延L_{IC}被用来保持同步输出。但是，在过程中BC的时延被设为δ_{out}，这个值是两倍时延，也就是$2L_{IC}$，因为这与全QR阵列的原始调度相关，因此这展示了没有哪两个连续的BC运算是在连续的周期上执行的。通过保持数据调度的结构，重定时过程实质上是一个简单的关系，见表12-3。

表12-2　IC一般的定时。摘自*Design of a Parameterizable Silicon Intellectual Property Core for QR-Based RLS Filtering*，by G. Lightbody & R. Woods，IEEE Trans on VLSI Systems，Vol. 11，No. 4，© 2003 IEEE

IC	Delay value
I_{RL}	$2P_A + P_M + P_R - T_{QR}$
PI1	$T_{QR} - P_A - P_S$
PI1a	如果$I_{RL} < 0$，则为$-I_{RL}$，其他为0
PI1b	如果$I_{RL} < 0$，则为PI1 − PI1a，其他为PI1
PI2	如果$I_{RL} > 0$，则为I_{RL}，其他为PI1
PI3	$PI2 + P_A + P_M$
PI4	$L_{IC} - L_x$
PI5	L_{IC}
PI6	$L_{IC} - PI2$

表12-3　BC和IC的时延的一般的表达。摘自*Design of a Parameterizable Silicon Intellectual Property Core for QR-Based RLS Filtering*，by G. Lightbody & R. Woods，IEEE Trans on VLSI Systems，Vol. 11，No. 4，© 2003 IEEE

时延	值
L_a	P_M
L_b	$P_M + PB5$
L_δ	$PB3 + PB4 + 2P_M$
L_x	PI3 + PA

12.8.1 重定时 QR 结构

接下来的这一节将继续讨论重定时问题，以及如何在一个一般的架构中包含它们。

线性阵列架构的重定时

时延有伸展每个 QR 更新运算调度的作用。这意味着迭代 $n=2$ 在迭代 $n=1$ 开始之后 $2m+1$ 个时钟周期起动。但是，处理器时延的引入扩展了调度表，使得 $n=2$ 的迭代在 $(2m+1)$ L_{IC}个时钟周期后起动。很明显这不是线性架构的最优化使用，因为它只能在每个 L_{IC}时钟周期被使用。

一个由 T_{QR}表示的因子，在最后一节作为连续的 QR 更新的开始周期的数量被引入，它是由硬件简化的程度决定的。可以看出导致一个 100% 利用率的有效调度，能通过设定时延 L_{IC}与 T_{QR}是相对质数来得到实现。也就是，如果两个值不共享一个除了 1 的共同因子，那么它们的最小公倍数会是它们的乘积。否则在 L_{IC}与 T_{QR}的乘积和它们的公倍数之间会有数据冲突。

如果　　$T_{QR} = \mathrm{mod}\quad c$　且　$L_{IC} = \mathrm{mod}\quad c$

然后　　$T_{QR} = d \times c$ 且 $L_{IC} = e \times c$

得

$$c = \frac{T_{QR}}{d} = \frac{L_{IC}}{e}$$

式中，c 是 T_{QR}和 T_{IC}的一个公倍数，是一个除了 1 以外的正整数，并且 d 和 e 分别是 L_{IC}和 L_{IC}的一个因子。因此，在 $T_{QR}\times e = L_{IC}\times d$ 中会有冲突。

这意味着 $T_{QR}\times e$ 和 $L_{IC}\times d$ 的乘积一定比 $T_{QR}\times L_{IC}$小。因此，会有冲突数据。相反，为了获得无冲突数据，就设 c 为 1。

时间例子 $T_{QR}\times L_{IC}$不代表一个数据冲突，因为 T_{QR}的值等于 $2m+1$，并且避免线上 QR 运算与一个新的 QR 运算冲突。在选择最优化值 T_{QR}和 L_{IC}的另一个重要因素是确保处理器 100% 的效率。

T_{QR}和 L_{IC}之间简单的关系在实现每个高利用率的类型结构中是一个关键因素。更重要的是，这个关系给予了一个简明的数学表达，这个表达在一个通用 QR 架构自动生成，通过调度来完成，并且在解决重定时的问题中是被需要的。

图 12-34 所示为在图 12-17 中展示的 7 输入线性阵列的一个调度的例子，其中 L_{IC}为 3 且 $T_{QR}=7$。阴影单元代表了来自不同更新的 QR 运算，这个更新与其余运算有交错并且这个阴影单元填补了由突出的 QR 更新留下的空位。这个调度被确认由第一个 QR 更新的数据所填充；因此，这是依靠 L_{IC}时延的。

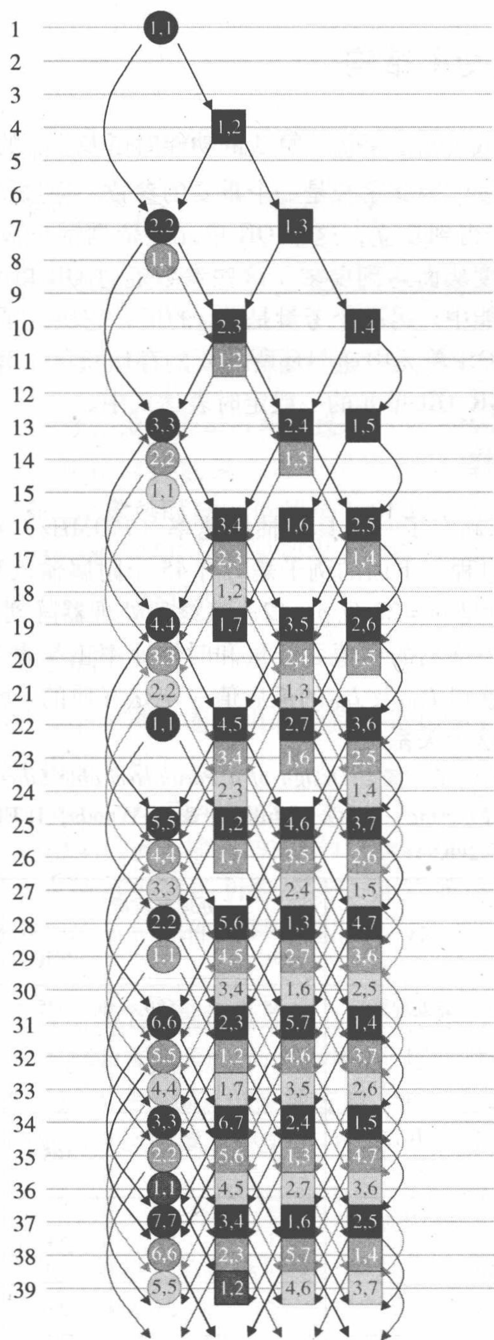


图 12-34　IC 时延为 3 的线性阵列的调度。摘自 *Linear QR Architecture for a Single Chip Adaptive Beamformer*, by G. Lightbody, R. Woods, R. Walke & J. McCanny, Journal of VLSI Signal Processing Systems 24, 67 – 81 2000, Kluwer Academic Publishers, Springer

12.9 可参数化 QR 结构

参数化的主要领域包括了字长、算法的功能时延及 T_{QR}的值。不同的规范会要求不同的有限精确度，因此字长是一个重要的参数。通过使用算法功能的一个分层库，QR 单元已经得到建立，这个 QR 单元是依据字长被参数化的，也可用流水线来增加运算速度从而达到要求。这些参数通过 QR 单元 HDL 描述的分层被传到这些算法的功能中。另一个考量是 T_{QR}的值，它决定了从一个 QR 的 R 和 u 的值更新到下一个 QR 单元中递归环所需要的存储长度。算法功能中的 T_{QR}和流水线的级被并入 SGR QR 单元的一般定时表达式中。

12.9.1 结构的选择

表 12-4 证实了当确定了一个具体抽样速率（100MHz）和 QR 阵列大小时，设计一个 QR 架构的过程。下面的例子是对有 45 个附属输入和 12 个主要输入的 QR 阵列而言的，也就是 $m=22$ 且 $p=12$。最终的处理器阵列是 $2m+1=45$ 行和 $m+p+1=35$ 列。只要给出的样值吞吐量和时钟速率正如表 12-4 所示，我们就能决定 T_{QR}的值。注意到 T_{QR}及 L_{IC}的最后值一定是互质的，但是对于这些例子，我们现在可以不考虑这个关系。

表 12-4 QR 架构的例子。摘自 *Design of a Parameterizable Silicon Intellectual Property Core for QR-Based RLS Filtering*, by G. Lightbody & R. Woods, IEEE Trans on VLSI Systems, Vol. 11, No. 4, © 2003 IEEE.

架构	处理器数量			T_{QR}	数据速率（MSPS）
	BC	IC	总共		
全 QR 阵列（每个 QR 单元都有处理器）	45	1530	1575	4	25
矩形阵列 1（12 个线性阵列，其中 4 行有一个）	12	408	420	4	25
矩形阵列 2（3 个线性阵列，其中 15 行有一个）	3	102	105	15	6.67
稀疏矩形阵列（3 个线性阵列的处理器且每个是 2 列 IC）	3	51	54	30	3.33
线性（1BC，26IC）	1	34	35	35	2.22
一个线性阵列上每个 IC 处理器上有稀疏线性 2 列 IC	1	17	18	18	1.11

T_{QR}的一般描述如上所示被重新排列，可以给出下面的关系：

$$\frac{N_{IC}}{N_{rows}}=\frac{T_{QR}}{2m+1}$$

这个结果被四舍五入到最近的整数。有三个可能：

$$\text{如果}\frac{T_{QR}}{2m+1}>1\text{，则需要一个稀疏线性阵列}$$

$$\text{如果}\frac{T_{QR}}{2m+1}=1\text{，则需要一个线性阵列}$$

$$\text{如果}\frac{T_{QR}}{2m+1}<1\text{，则需要一个矩形阵列}$$

依靠结果架构的维度，设计者可以决定选择一个稀疏矩形架构。注意到全三角阵列能满足一个 100MHz 时钟的最大吞吐量被限制在 25MSPS 内，这是因为 QR 单元递归路径中有 4 周期的时延（也就是 100MHz ÷ 4 = 25MSPS）。当第一个矩形阵列方案使用 408IC 和 12BC 而不是要求 1530IC 和 45BC 的全阵列时，可以满足与全 QR 阵列相同的吞吐量性能。

12.9.2　可参数化控制

不同架构设计的一个关键方面是由驱动这些结构中乘法器的控制数据决定的。因为已经得到应用的各种映射，所以认为 IC 运算有 4 个不同的运算模式是更贴切的，分别是输入、反射的输入、未反射输入和反射单元（见图 12-35，Walke1997），其中 x 代表 x 输入的数据且代表了旋转参数 θ。反射的 IC 是从 QR 阵列导出矩形处理器阵列的折叠结果，并且简单地反映出不同的数据流。单元的定位是由乘法器和控件共同管理的，因此单元的定位是一个关于控制信号发生的问题。

运算的 4 个模式通过两个控制信号控制，其中一个是 C，它决定 x 是来自阵列（I）的还是来自外部数据（E）的；另外一个是 F，它是区别折叠（M）和非折叠（U）的运算。后者决定输入的资源方向。输出来自单元的对面。每个架构中决定控制部分的机制在后面章节中给出。

12.9.3　线性架构

线性架构的控制信号源自它的数据调度。单元的运算模式在每个调度周期被决定，见表 12-5。图 12-36 所示需要有用来接收和处理正确数据的环境控制和乘法器的 QR 单元。表 12-6 给出了一个全 QR 运算的控制信号的例子。

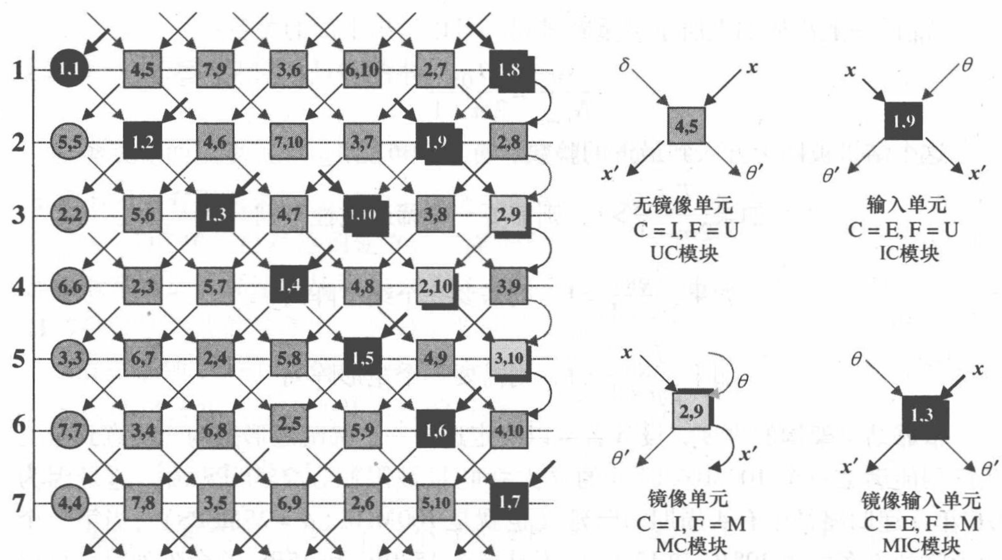


图 12-35 处理器阵列中的单元种类。摘自 *Design of a Parameterizable Silicon Intellectual Property Core for QR-Based RLS Filtering*, by G. Lightbody & R. Woods, IEEE Trans on VLSI Systems, Vol. 11, No, 4, © 2003 IEEE

表 12-5 线性阵列的 QR 单元的运算的模式

周期	BC_1	IC_2	IC_3	IC_4	IC_5	IC_6	IC_7
1	IC	UC	UC	UC	UC	UC	MIC
2	UC	IC	UC	UC	UC	MIC	UM
3	UC	UC	IC	UC	MIC	UC	MIC
4	UC	UC	UC	IC	UM	MIC	UM
5	UC	UC	UC	UC	IC	UM	MIC
6	UC	UC	UC	UC	UM	IC	UM
7	UC	UC	UC	UC	UM	UM	IC

其他变式架构的控制和定时更加复杂，但是都源自原始线性阵列的控制。需要考虑控制序列上时延的影响，并且在矩形的变式中，单元需要能够接收来自上下单元和邻近单元的输入 x 和 θ。每个变式应该被轮流监控。

12.9.4 稀疏线性架构

图 12-37a 所示为被分配到每个 IC 上的两列运算。从图 12-37b 所示的部分调度中可以看出，一个在阵列中从左到右的值要进行传递需要大量的时延。从 BC（1，1）到邻近的 IC（1，2）的传递 $\boldsymbol{\theta}_1$ 花费了 3 个周期。但是，从 IC 到 BC

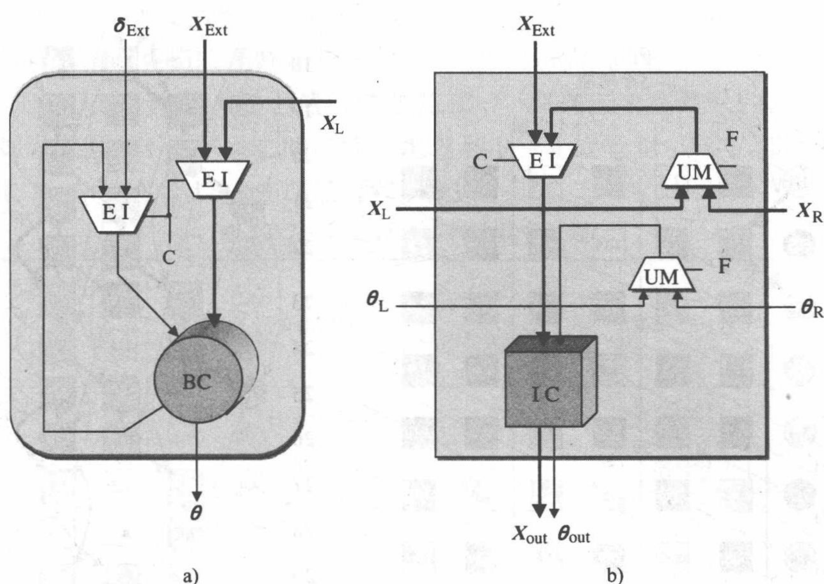


图 12-36 线性架构的 QR 单元。摘自 *Design of a Parameterizable Silicon Intellectual Property Core for QR-Based RLS Filtering*, by G. Lightbody & R. Woods, IEEE Trans on VLSI Systems, Vol. 11, No. 4, © 2003 IEEE

a）边界单元 b）内部单元

的传递 X_{12} 仅仅花费了 1 个周期。

表 12-6 外部/内部 *x* 输入和反射/非反射单元的线性阵列控制

周期	C_1	C_2	C_3	C_4	C_5	C_6	C_7	F_1	F_2	F_3	F_4	F_5	F_6
1	E	I	I	I	I	I	I	U	U	U	U	U	M
2	I	E	I	I	I	I	I	U	U	U	U	M	U
3	I	I	E	I	I	I	I	U	U	U	M	U	M
4	I	I	I	E	I	I	I	U	U	U	U	M	U
5	I	I	I	I	E	I	I	U	U	U	U	U	M
6	I	I	I	I	I	E	I	U	U	U	U	U	U
7	I	I	I	I	I	I	E	U	U	U	U	U	U
8	E	I	I	I	I	I	E	U	U	U	U	U	M
9	I	E	I	I	I	E	I	U	U	U	U	M	U
10	I	I	E	I	E	I	I	U	U	U	M	U	M
11	I	I	I	E	I	I	I	U	U	U	U	M	U
12	I	I	I	I	E	I	I	U	U	U	U	U	M
13	I	I	I	I	I	E	I	U	U	U	U	U	U

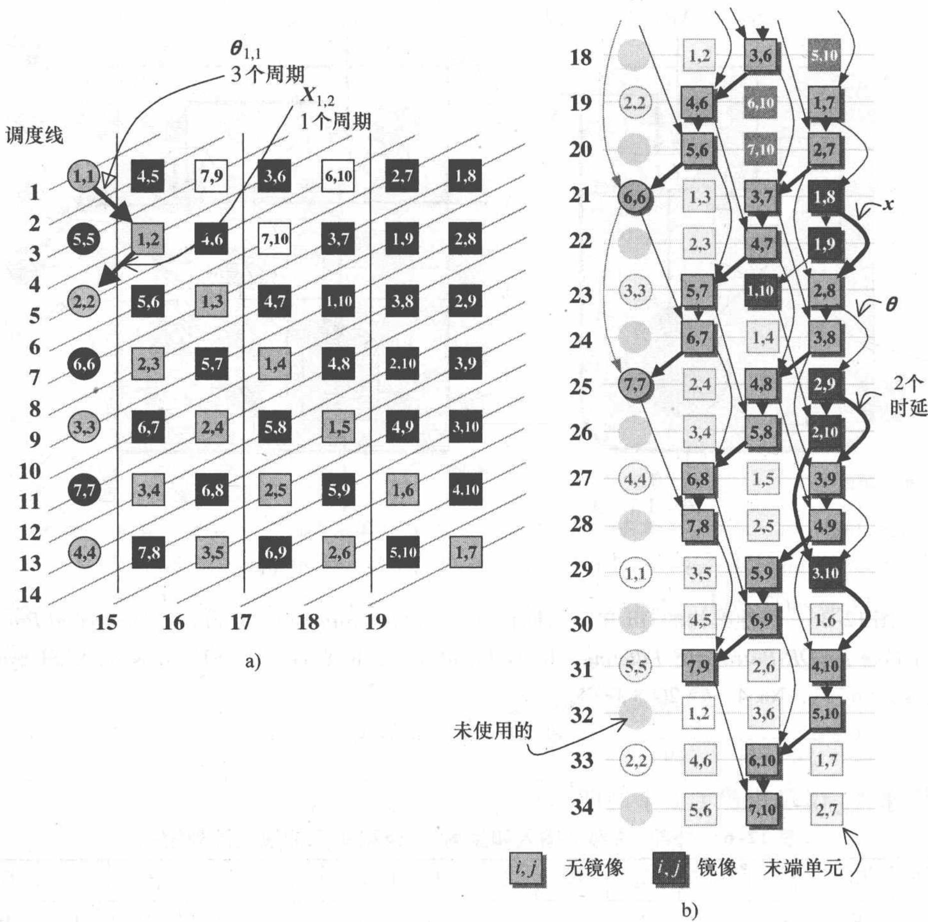


图 12-37 稀疏的线性阵列调度

a）处理器阵列 b）调度

图 12-38 中的例子展示了 IC 的 3 列的分区。将它们调度到一个单个的处理器上，要求维持住它们的相继次序。IC 运算已经被编号了，第 1 行是 1，2 和 3；第 2 行是 1′，2′和 3′。从运算 2 产生的输出为 1′和 3′。因为所有的运算都在同一个处理器上执行，所以需要保留这些时延值，一直保留到运算 1′和 3′需要它们时。运算 3 在运算 1′之前被执行，并且运算 3，1′和 2′都在运算 3′之前执行，这分别与两个和 4 个周期的时延有关。

根据如图 12-39 所示被分配到每个 IC 上的处理器阵列中运算的列的数量 N_{IC}，这已经得到了一般的定义。

一般来说，x 和这两个输出值被从运算（$c+1$）传递到 c 和（$c+2$）中来匹

配图 12-39。这个被反馈到一个具体的运算上的值，依靠表 12-6 中所概括的来判断执行运算是折叠模式还是非折叠模式。如果数据在同类型的单元之间传输（也就是 U→U，或者 M→M），然后根据表 12-7，则时延要么是 -1 要么是 +1。但是，如果数据在不同单元之间传输（也就是 U→M 或者 M→U），那么时延的数量会是 N_{IC}。这在表 12-7 中已经得到了概括。

表 12-7　稀疏线性阵列的要求时延

数据在单元之间的传递： U：不反射 M：反射的	按照 QR 运算的方向	在处理器阵列中显示的数据流的方向	需要的时延	标号
U→U	$(i, j) \rightarrow (i, j+1)$，$\theta$	→	$N_{IC}+1$	D_1
	$(i, j) \rightarrow (i+1, j)$，$x$	←	$N_{IC}-1$	D_2
M→M	$(i, j) \rightarrow (i+1, j)$，$\theta$	←	$N_{IC}-1$	D_2
	$(i, j) \rightarrow (i, j+1)$，$x$	→	$N_{IC}+1$	D_1
U→M（最后的单元）	$(i, j) \rightarrow (i, j+1)$，$\theta$	↓	N_{IC}	D_3
M→U（最后的单元）	$(i, j) \rightarrow (i+1, j)$，$x$	↓	N_{IC}	D_3

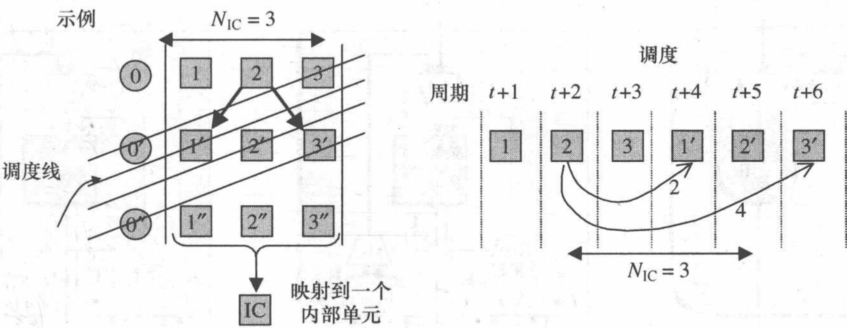


图 12-38　一个处理器上的三列的分区的例子

在稀疏线性架构中，这些时延可以保持想要的调度，如图 12-40 所示。时延的三个级恰好被标为 D_1，D_2 和 D_3。这些时延能被重分配来形成一个更加有效率的 QR 单元架构，如图 12-41 所示。附加的 L 和 R 控制信号表明输入的方向源，而 E 和 I 控制值决定输入是来自一个邻近的单元还是同样的单元。EC 指的是最后略微不同的 IC，因为当单元需要从输出接收输入时，会有两个运算模式。这个例子的控制序列见表 12-8。

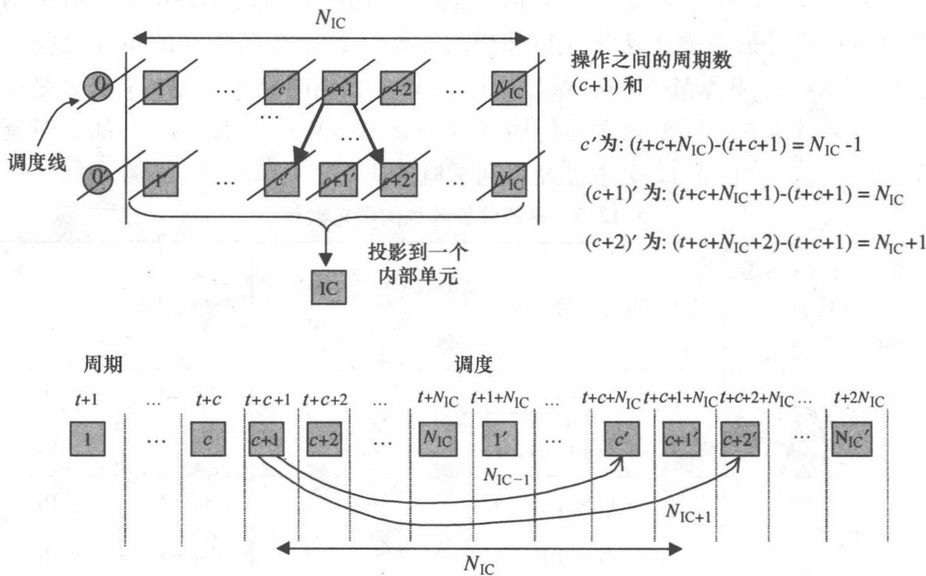


图 12-39 N_{IC}列投影到处理器上的通用分区

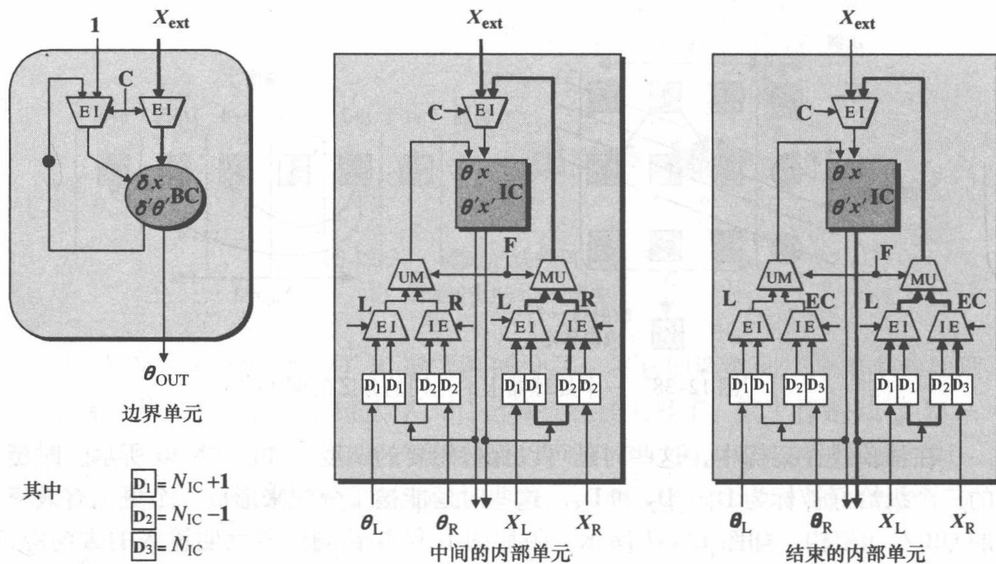


图 12-40 稀疏线性阵列单元

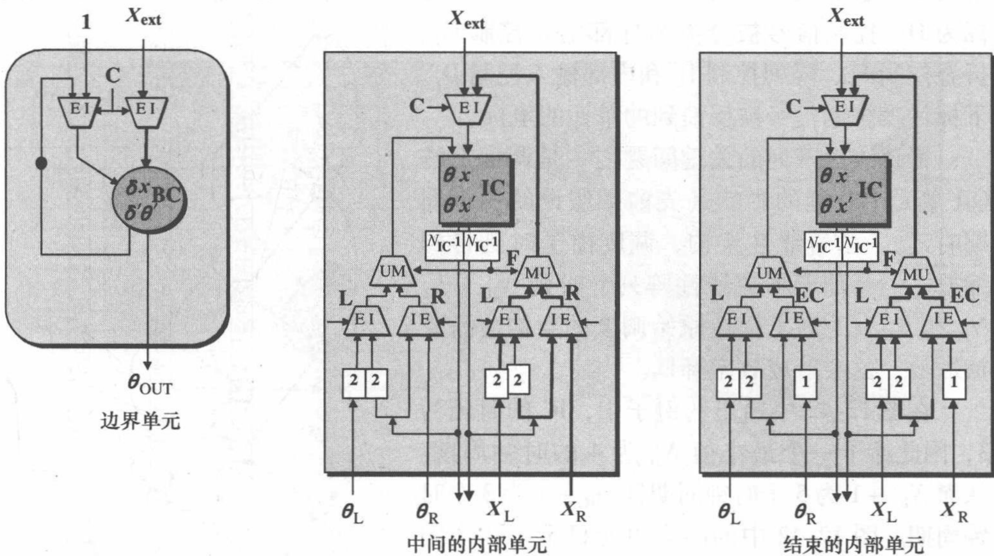


图 12-41　分布时延的稀疏线性阵列单元

表 12-8　稀疏线性阵列的控制序列

周期	外部输入	外部输入控制				折叠控制			内部控制		
		C_1	C_2	C_3	C_4	F_2	F_3	F_4	L	R	EC
1	$X_1(1)$	E	I	I	I	U	U	M	E	I	E
2	$X_8(0)$	I	I	I	E	U	U	U	I	E	I
3		I	I	I	I	U	U	U	E	I	E
4	$X_2(1)$	I	E	I	I	U	U	U	I	E	I
5	$X_7(0)$	I	I	I	E	U	U	U	E	I	E
6		I	I	I	I	U	U	U	I	E	I
7	$X_3(1)X_8(0)$	I	E	I	E	U	U	M	E	I	E
8	$X_9(0)$	I	I	I	E	U	U	M	I	E	I
9	$X_{10}(0)$	I	I	E	I	U	M	U	E	I	E
10	$X_4(1)$	I	I	E	I	U	U	U	I	E	I
11		I	I	I	I	U	U	M	E	I	E
12		I	I	I	I	U	U	M	I	E	I
13	$X_5(1)$	I	I	E	I	U	U	U	E	I	E
14		I	I	I	I	U	U	U	I	E	I

从表 12-8 中，可以看出 EC 与 R 相同而与 L 相反。此外，由于 E 和 I 的每

个周期状态的交替，因此一个控制序列可以决定内部输入的控制。这个控制值被标为 D。控制信号被分类为外部输入控制 C_i、折叠控制 F_i、阵列控制 L_i 和内部输入控制 D_i。下标代表控制信号被反馈到的单元的坐标。

稀疏线性阵列的关键问题之一是调度上的 QR 单元中时延的影响（先前被假设的一个周期时延）。随着线性架构，调度由于时延得到缩放。但是随着稀疏线性阵列，时延 $N_{IC}-1$，N_{IC} 和 $N_{IC}+1$ 为了保持原始调度的架构会需要被缩放，这会造成效率降低。

在图 12-42 中给出的例子中，IC 的时延为 3，因此给予一个最小值 N_{IC} 为 4 的时钟周期。从而 $N_{IC}+1$ 为 5 个时钟周期且 $N_{IC}-1$ 为 3 个时钟周期。图 12-42 中的阴影单元显示了一个完整的有着互连性的 QR 更新。剩下的 QR 运算被展示出来，但是限制细节变得简洁了。IC 的时延可能会超过被分配到每个处理器上的列的数量，稀疏线性阵列中的时延会依靠 L_{IC} 就讲得通了，也就是时延($N_{IC}-1$)不会被需要并且调度的重排会被执行一个或两个周期时延，这个时延在图 12-41 中得到了编号。突出的单元代表了一个全 QR 的更新，同时另一个被标号的单元代表了交错的 QR 运算。调度中没有被标号的灰色 BC 代表没有被使用的位置。

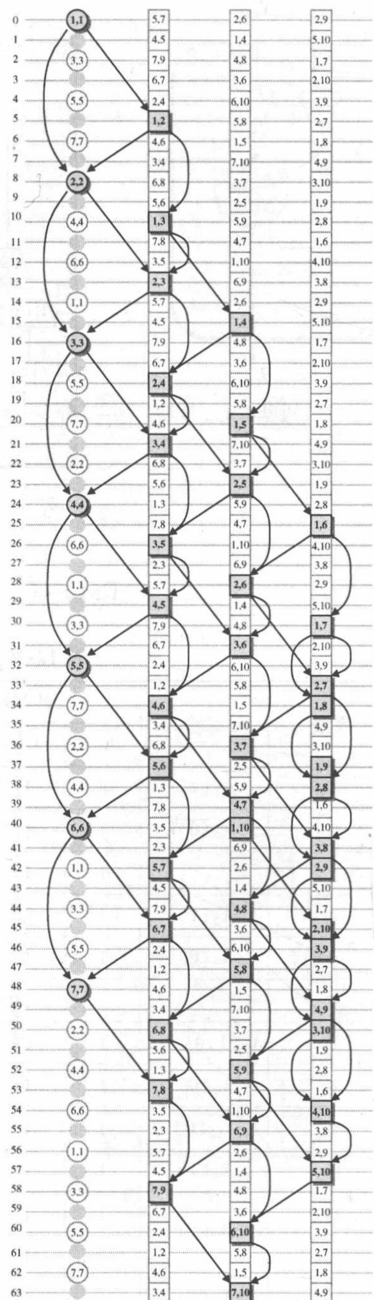


图 12-42　稀疏线性阵列在调度时延上的影响（$L_{IC}=3$）

12.9.5　矩形架构

矩形架构包含了串联的多重线性阵列架构。因此，需要配置 QR 单元来使得它们能够从上面的线性阵列接收输入。此外，顶部的线性阵列需要能够从底部的线性阵列接收值。QR 单元如图 12-43 所示。控制信号 E 和 I 决定 X 输入是外部的（也就是系统输入）还是内部的。控制值 T 指来自上面阵列的输入且 A 指来自邻近单元的输入。当被作为下标时，TR 和 TL 指来自上面阵列的右单元和左单元的

值；AR 和 AL 指同样的线性阵列中来自右和左邻近单元的值。

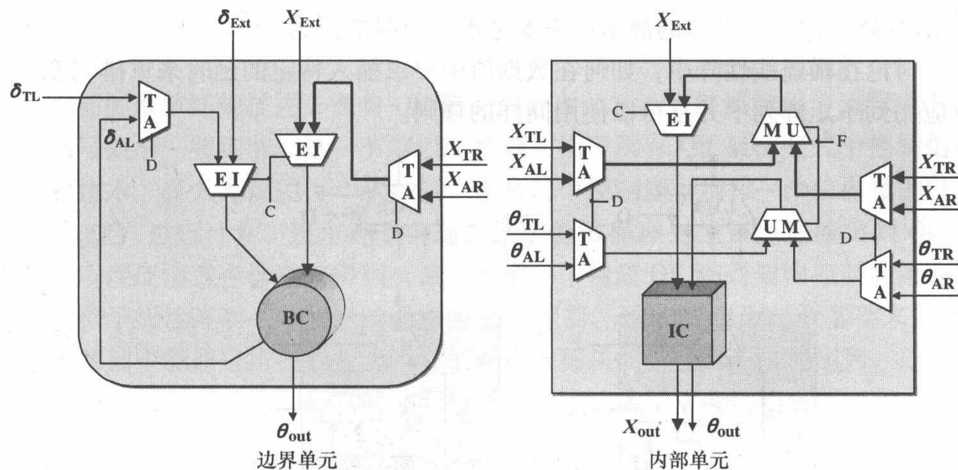


图 12-43　矩形阵列的 QR 单元

12.9.6　稀疏矩形架构

稀疏矩形阵列的 QR 单元需要能反馈输入到它们自身中，已经讨论了线性构架和矩形架构在此部分的变换。附加的控制电路被包含在如图 12-44 所示的 QR

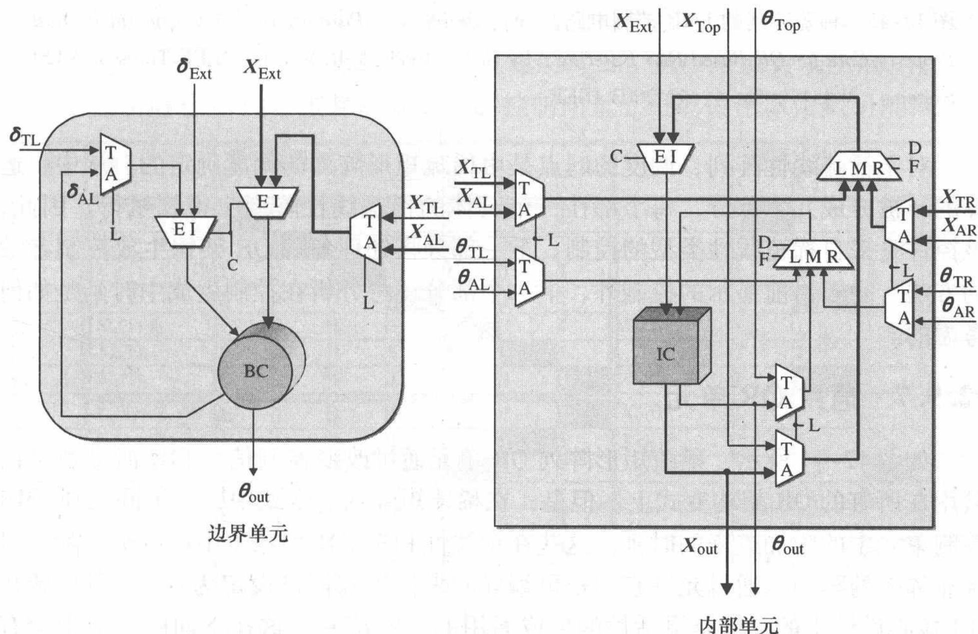


图 12-44　稀疏矩形阵列的 QR 单元。摘自 *Design of a Parameterizable Silicon Intelletual Property Core for QR-Based RLS Filtering*，by G. Lightbody & R. Woods，IEEE Trans on VLSI Systems，Vol. 11，No. 4，© 2003 IEEE

图中。控制和时延想要通过稀疏阵列的重排调度被一起放进 LMR 复用器单元（图 12-45）内，在本节的演示中需要考虑重定时的延迟。

讨论在稀疏线性阵中，如何在数据值中要求插入特定的延时来重排调度。这也应用到了矩阵列中并且可以使用同样的规则。

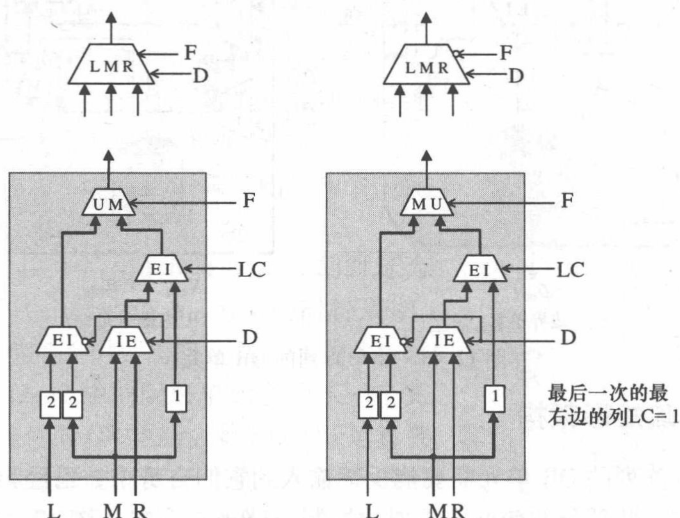


图 12-45　稀疏阵列的 LMR 控制电路。摘自 *Design of a Parameterizable Silicon Intellectual Property Core for QR-Based RLS Filtering*, by G. Lightbody & R. Woods, IEEE Trans on VLSI Systems, Vol. 11, No. 4, © 2003 IEEE

对于稀疏线性阵列，调度的起点是由稀疏矩形阵列的调度确定的。从中，运算的行被分成几个部分，每个都在一个具体的稀疏线性阵列上得到执行。因此，该控件是来自稀疏线性类型的控制。下一部分将处理各种 QR 架构生成控制参数的方法。除了目前显示的控制外，下一个部分还将分析在控制生成中时延是如何考虑的。

12.9.7　通用 QR 单元

如图 12-44 所示，稀疏矩形阵列 QR 单元通过改变控制信号和定时参数可以被用在所有的 QR 架构变式上。但是，在稀疏矩阵列的变式中，有被嵌入到 LMR 控制单元中的附加进来的时延。这些在全线性和矩形阵列版本中可以被移除，对于非稀疏的阵列，通过允许它们是可编译的来使得它们被设定为 0。在可参数化 QR 核心设计中的关键是灵活性的生成通用的控制信号。这在下面的章节中会有讨论。

12.10 通用控制

先前的章节详细描述了源自 QR 阵列的各种架构。对需要被用于运行电路的控制信号给出一些细节。这一节着眼于能被应用于所有 QR 架构变式中控制信号的一般技术。有人提议用一个软件接口来将每个控制序列当作一个位向量的种子（如 T_{QR}值）进行计算，这个种子会通过这个值被循环一位一位地输出到 QR 单元中的线性反馈寄存器中来得到反馈。发展一个稀疏 QR 阵列架构控制的第一步是考虑给予线性阵列一个所需的通用控制处理器。这能被作为一个基础来使用，从这个基础中能通过折叠和操纵控制信号进入稀疏线性阵列要求的序列，以此来决定控制。矩阵列的控制会从线性架构的控制中被简单快速地生成。

12.10.1 线性与稀疏线性阵列的通用输入控制

外部输入的 **X** 值跟随有新的外部输入 **X** 的折叠 QR 三角阵列进入到每个周期线性阵列的单元中。从最左边的单元开始，一直到最右边的单元，然后折叠回来直到所有的 $2m+p+1$ 个输入被反馈到具体的 QR 更新的阵列中。其中图 12-46 所示的一组 QR 输入得到突出。下一组输入跟随同一路径反馈，但是是在周期 T_{QR}之后开始的。这就导致每周期重复一次（对线性阵列例子来说是 7；对稀疏线性阵列例子来说是 14）地控制一个重复的部分。从中可能自动生成 T_{QR}，代表每个控制信号 $C_1 \sim C_7$ 重复部分的位的控制向量。可以使用原始数组的维数来

周期	输入	线性阵列							输入	稀疏线性阵列			
		C_1	C_2	C_3	C_4	C_5	C_6	C_7		C_1	C_2	C_3	C_4
1	$X_1(1), X_8(0)$	E	I	I	I	I	I	E	$X_1(1)$	E	I	I	I
2	$X_2(1), X_9(0)$	I	E	I	I	I	E	I	$X_6(0)$	I	I	I	E
3	$X_3(1), X_{10}(0)$	I	I	E	I	E	I	I		I	I	I	I
4	$X_4(1)$,	I	I	I	E	I	I	I	$X_2(1)$	I	E	I	I
5	$X_5(1)$,	I	I	I	I	E	I	I	$X_7(0)$	I	I	I	E
6	$X_6(1)$,	I	I	I	I	I	E	I		I	I	I	I
7	$X_7(1)$,	I	I	I	I	I	I	E	$X_3(1), X_8(0)$	I	E	I	E
8	$X_8(1), X_1(2)$	E	I	I	I	I	I	E	$X_9(0)$	I	I	I	E
9	$X_9(1), X_2(2)$	I	E	I	I	I	E	I	$X_{10}(0)$	I	I	E	I
10	$X_{10}(1), X_3(2)$	I	I	E	I	E	I	I	$X_4(1)$	I	I	E	I
11	$X_4(2)$	I	I	I	E	I	I	I		I	I	I	I
12	$X_5(2)$	I	I	I	I	E	I	I		I	I	I	I
13	$X_6(2)$	I	I	I	I	I	E	I	$X_5(1)$	I	I	E	I
14	$X_7(2)$	I	I	I	I	I	I	E		I	I	I	I

图 12-46　线性 QR 阵列的外部输入的控制。摘自 *Design of a Parameterizable Silicon Intellectual Property Core for QR-Based RLS Filtering*，by G. Lightbody & R. Woods，IEEE Trans on VLSI Systems，Vol. 11，No. 4，© 2003 IEEE

实现确定一系列 QR 输入的开始与之前设置的开始的关系，由此产生的处理器阵列生成了简化的 QR 架构。这个关系如图 12-47 所示，粗线表示一个 QR 更新的一系列输入。图 12-46 突出展示了线性阵列的外部输入的控制。

用软件代码写出生成线性阵列和稀疏线性阵列的外部输入的控制信号。输入被分解成两部分（见图 12-47），一个处理从左到右的输入，而另一个处理从右到左的输入（方向的改变是由于折叠造成的）。代码生成了每个输入到每个处理器的控制向量中控制信号的位置。如果输入的一个向量的模大于另一个向量，那么两个向量相减，由留下的模数确定起始位置。但是，在 QR 阵列的最初的运算之后，有必要通过一个适当的值来延时这个控制信号。

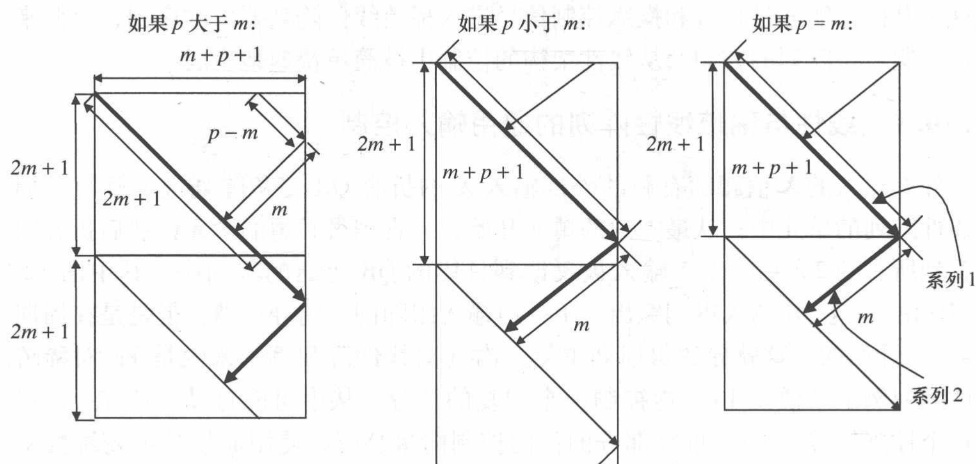
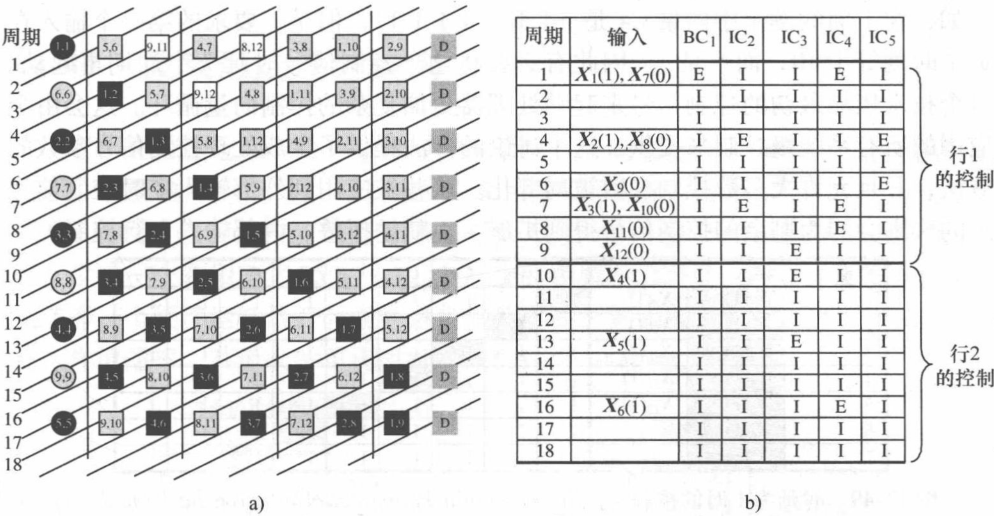


图 12-47 线性 QR 阵列的外部输入的控制。摘自 *Design of a Parameterizable Silicon Intellectual Property Core for QR – Based RLS Filtering*, by G. Lightbody & R. Woods, IEEE Trans on VLSI Systems, Vol. 11, No. 4, © 2003 IEEE

12.10.2 矩形与稀疏矩形阵列的通用输入控制

来自矩形阵列的控制相当简单，并且是直接来自线性阵列的控制向量，如图 12-46 所示，通过将信号向量分成几个与处理器阵列中分区有关的部分。例如，如果线性阵列的控制种子向量是 8 位宽且相同系统的矩形阵列由两行组成，那么每个控制向量种子会被分成两个 4 位宽的向量。其中一个生成第一个矩阵列，另一个生成第二个矩阵列。稀疏矩阵列的控制种子以同一种方式得到派生，由有着一样 N_{IC} 值的稀疏线性阵列控制。同样的代码可被编译纳入稀疏阵列需要的虚拟运算。图 12-48a 所示为一个 $m=4$，$p=3$ 且 $N_{IC}=2$ 的稀疏线性阵列映射的实例。这个控制（见图 12-48b）能被分成实现一个由稀疏线性阵列的两行组成的稀疏矩形阵列的两个部分。



周期	输入	BC_1	IC_2	IC_3	IC_4	IC_5
1	$X_1(1), X_7(0)$	E	I	I	E	I
2		I	I	I	I	I
3		I	I	I	I	I
4	$X_2(1), X_8(0)$	I	E	I	I	E
5		I	I	I	I	I
6	$X_9(0)$	I	I	I	I	E
7	$X_3(1), X_{10}(0)$	I	E	I	E	I
8	$X_{11}(0)$	I	I	I	E	I
9	$X_{12}(0)$	I	I	E	I	I
10	$X_4(1)$	I	I	E	I	I
11		I	I	I	I	I
12		I	I	I	I	I
13	$X_5(1)$	I	I	E	I	I
14		I	I	I	I	I
15		I	I	I	I	I
16	$X_6(1)$	I	I	I	E	I
17		I	I	I	I	I
18		I	I	I	I	I

图 12-48　控制种子的分区。摘自 *Design of a Parameterizable Silicon Intellectual Property Core for QR-Based RLS Filtering*, by G. Lightbody & R. Woods, IEEE Trans on VLSI Systems, Vol. 11, No. 4, © 2003 IEEE

a）稀疏阵列实例　b）外部输入控制

12.10.3　时延对控制种子的影响

下一步是确定时延在控制向量上的影响。先前的时延值 D_1，D_2 和 D_3，正如之前讨论的稀疏线性阵列，被应用到每个 IC 运算的多重列（N_{IC}）上很有必要，其中对于一个有单个周期时延的系统，$D_1=(N_{IC}-1)$，$D_2=N_{IC}$，$D_3=(N_{IC}+1)$。但是，在真正的系统中，处理器有多重时延。假设 IC(L_{IC})时延比这些时延都长，那么增加这些时延能使得合适的时延变为 $D_1=L_{IC}$，$D_2=L_{IC}+1$ 且 $D_3=L_{IC}+2$。对于线性阵列，D_1，D_2 和 D_3 被设为 L_{IC}，然后用编码生成控制向量。唯一的不同是当控制值位置超过了向量宽度的时候。和单个时延版本一起，这通过从值中减去 T_{QR} 值来解释（其中向量种子的宽度是 T_{QR}）。

时延控制序列的重复部分是复杂的。但是，长度为 T_{QR} 的一个重复的控制序列仍然能找得到。当时延被包含进计算时，通过减小值来达到向量宽度的边界是不够的。或者，向量中控制值的位置通过 T_{QR} 的模数来找到。通过一个线性阵列的例子来展示对控制向量上的时延影响的分析，其中这个阵列的 $m=3$，$p=5$ 且 $T_{QR}=2m+1=7$。图 12-49 所示为有单个周期时延系统的控制向量。图 12-50 所示为当时延为 3 时一个 QR 更新的输入集合控制定时的影响。通过使用这些控制值位置，给出了如图 12-51 所示的控制向量。

要强调的一点是，在被要求的输入出现之前也许会有控制向量的几个周期。

例如，在上面的例子中向量 C4 是 [I I E I I I I]，但是，要求的第一个输入在那个时刻等于 10，而不是 3。因此有必要将这个控制信号时延 7 个周期再起动。这个技术依靠最初的控制信号来起动处理器更加复杂的控制向量循环。这会由全应用的所有系统级控制来提供。这个讨论的方法提供了生成处理控制信号参数的方法，并且允许大多数控制参数被局部化。这些被应用到外部输入定时控制信号上的法则会因为剩余的控制信号得到扩展，也就是折叠、内部输入和行控制。

周期	输入		C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9
1	$X_1(1)$	$X_8(1)$	E	I	I	I	I	I	I	E	I
2	$X_2(1)$	$X_9(1)$	I	E	I	I	I	I	I	I	E
3	$X_3(1)$	$X_{10}(1)$	I	I	E	I	I	I	I	I	E
4	$X_4(1)$	$X_{11}(1)$	I	I	I	E	I	I	I	E	I
5	$X_5(1)$	$X_{12}(1)$	I	I	I	I	E	I	E	I	I
6	$X_6(1)$		I	I	I	I	I	E	I	I	I
7	$X_7(1)$		I	I	I	I	I	I	E	I	I

图 12-49　时延为 1 时的控制。摘自 *Design of a Parameterizable Silicon Intellectual Property Core for QR-Based RLS Filtering*, by G. Lightbody & R. Woods, IEEE Trans on VLSI Systems, Vol. 11, No. 4, © 2003 IEEE

输入	时延 = 1		时延 = 3	
	时间	位置矢量	时间	位置矢量
X_1	1	1	1	1
X_2	2	2	4	4
X_3	3	3	7	7
X_4	4	4	10	10mod7 = 3
X_5	5	5	13	13mod7 = 6
X_6	6	6	16	16mod7 = 2
X_7	7	7	19	19mod7 = 5
X_8	8	8mod7 = 1	22	22mod7 = 1
X_9	9	9mod7 = 2	25	25mod7 = 4
X_{10}	10	10mod7 = 3	28	28mod7 = 0, 设置为7
X_{11}	11	11mod7 = 4	31	31mod7 = 3
X_{12}	12	12mod7 = 5	34	34mod7 = 6

图 12-50　决定控制向量中的位置。摘自 *Design of a Parameterizable Silicon Intellectual Property Core for QR-Based RLS Filtering*, by G. Lightbody & R. Woods, IEEE Trans on VLSI Systems, Vol. 11, No. 4, © 2003 IEEE

周期	输入				C_1	C_2	C_3	C_4	C_5	C_6	C_7	C_8	C_9
1		$X_8(1)$		$X_1(4)$	E	I	I	I	I	I	I	E	I
2		$X_6(2)$			I	I	I	I	I	E	I	I	I
3	$X_{11}(0)$		$X_4(3)$		I	I	I	E	I	I	I	E	I
4		$X_9(1)$		$X_2(4)$	I	E	I	I	I	I	I	I	E
5		$X_7(2)$			I	I	I	I	I	I	E	I	I
6	$X_{12}(0)$		$X_5(3)$		I	I	I	I	E	I	E	I	I
7		$X_{10}(1)$		$X_3(4)$	I	I	E	I	I	I	I	I	E

图 12-51　控制种子上时延的影响。摘自 *Design of a Parameterizable Silicon Intellectual Property Core for QR-Based RLS Filtering*, by G. Lightbody & R. Woods, IEEE Trans on VLSI Systems, Vol. 11, No. 4, © 2003 IEEE

12.11 波束形成设计实例

对于在雷达应用中的一个典型波束形成，m 的值会在 20 ~ 100。对于同样的应用，主要输入 p 的数量一般会在 1 ~ 5。图 12-52 所示为个范例。一个方式是使用 QR 阵列。假设最快的可能时钟速率为 f_{CLK}，然后基本的循环会决定范例的性能，生成一个利用率为 25% 且 T_{QR} 为 4 个时钟周期的设计。因此现在主要的难点是选择一个硬件运转最好时吞吐量最好的架构。例如这里，需要一个 15MSPS 的输入样值速率，其最大的可能时钟速率假设为 100MHz。

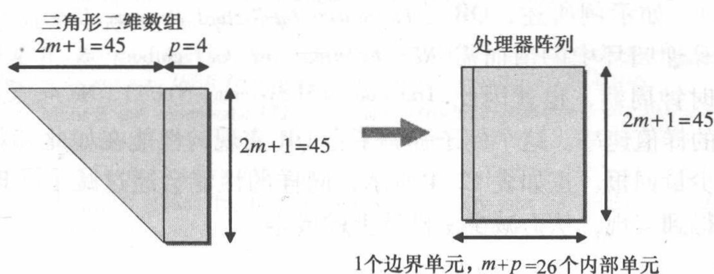


图 12-52 $m=22$，$p=4$ 时 QR 架构的推衍。摘自 *Design of a Parameterizable Silicon Intellectual Property Core for QR – Based RLS Filtering*，by G. Lightbody & R. Woods，IEEE Trans on VLSI Systems，Vol. 11，No. 4，© 2003 IEEE

T_{QR} 的值通过使用需要的样值速率 S_{QR} 和最大时钟速率 f_{CLK} 得到计算：

$$T_{QR}=\frac{f_{CLK}}{S_{QR}}=\frac{100\times10^6}{15\times10^6}=6.67$$

这个值是在连续的 QR 更新的起始之间被允许的最大周期数，因此，需要被四舍五入到最近的整数。N_{rows}/N_{IC} 的比率通过替代所知参数到下面的关系中能够得到计算：

$$\frac{N_{rows}}{N_{IC}}=\frac{2m+1}{T_{QR}}=\frac{45}{6}=7.5$$

其中，$1\leqslant N_{rows}\leqslant 2m+1$（也就是 45）且 $1\leqslant N_{IC}\leqslant m+p$（也就是 26）。通过使用这些指针，再将 N_{IC} 设为 2，N_{rows} 设为 15，一个有效率的架构就能够得到推衍，并且因此运算被分配到 15 个稀疏线性架构上，每个架构上有 1 个 BC 和 13 个 IC，如图 12-53 所示。

需要注意的是一定要考虑电路中的关键路径，以确保核心计算速度足够快，从而可以支持想要的 QR 运算。在这里，会添加额外的流水线站来减少关键路径并且因此提高时钟速率。但是，这些在架构分析中会对提高时延有影响。处理器

的每行要对处理器阵列中运算的 3 行负责，因此，$T_{QR}=6$，产生一个 16.67MSPS 的输入样值速率，它超过了要求的性能。在表 12-4 中给出了一些 QR 阵列架构的细节。

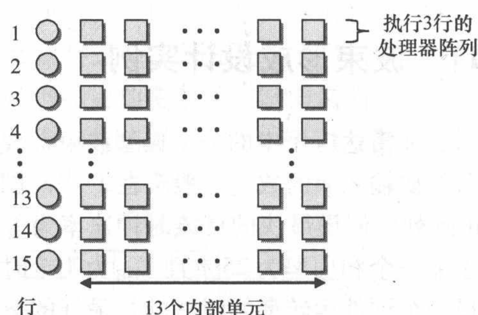


图 12-53 例子架构。摘自 *Design of a Parameterizable Silicon Intellectual Property Core for QR – Based RLS Filtering*, by G. Lightbody & R. Woods, IEEE Trans on VLSI Systems, Vol. 11, No. 4, © 2003 IEEE

全 QR 阵列实现的 T_{QR} 值由 QR 单元的递归环中的时延决定（由一个浮点加法和一个移位减法功能组成）。如示例所述，QR 阵列因为计算递归环中的值而需要等待 4 个时钟周期，也就因此决定了系统的样值速率。这个例子强调了全 QR 实现的性能在如此高花销的硬件情况下只有少量回报。正如表 12-4 所示，同样的性能会通过减小处理器数量的矩形阵列来得到实现，从而减少了硬件上的成本。

12.12 总结

本章的目标是记录自适应波束形成器的一个 IP 核发展的每一步。主要涵盖的方面是关于以下做出的设计选择：

1）发展一个 IP 核再用策略使用设计的决定；

2）算法的决定；

3）像一个 IP 核一样地设计一个合适组件的决定；

4）具体的架构参数；

5）可放缩架构；

6）运算和控制的可放缩调度。

以上列出的是详细的自适应波束形成器例子的每个阶段。背景资料提供的相关 RLS 算法决定采用自适应权值计算。使用算法的关键问题是它的计算复杂度。从技术和背景研究方面进行了综述，并且显示了一个适合在三角脉动阵列上实现被简化的 QR – RLS 算法的推衍。即使复杂度有减小，但是仍然需要映射全 QR 阵列到一套简化的架构上。给出一个通用设计是如何一步步得到实现过程的概述是本章的关键组成部分。注意力主要集中在架构的可放缩性及运算调度的影响上。所有调度问题上的处理器时延和定时的影响得到了进一步的详细描述，给出了这些因素的重要解释。最后，给出了对控制电路如何平衡架构发展的例子，同时给出维持性能的标准。按照设想本章包

括的原则应该被扩展到其他的 IP 核发展中。

参考文献

Athanasiadis T, Lin K and Hussain Z (2005) Space-time OFDM with adaptive beamforming for wireless multimedia applications. *ICITA 2005, 3rd Int. Conf. Information Technology and Applications*, pp. 381–386.

Baxter P and McWhirter J (2003) Blind signal separation of convolutive mixtures. *Conference Record of the 37th Asilomar Conference on Signals, Systems and Computers*, pp. 124–128.

Bitmead R and Anderson B (1980) Performance of adaptive estimation algorithms in dependent random environments. *IEEE Transactions on Automatic Control*, 25(4), 788–794.

Choi S and Shim D (2000) A novel adaptive beamforming algorithm for a smart antenna systemin a CDMA mobile communication environment. *IEEE Transactions on Vehicular Technology*, 49(5), 1793–1806.

Cioffi J (1990) The fast Householder filters-RLS adaptive filter. *ICASSP-90. International Conference on Acoustics, Speech, and Signal Processing*, pp. 1619–1622.

Cioffi J and Kailath T (1984) Fast, recursive-least-squares transversal filters for adaptive filtering. *Acoustics, Speech, and Signal Processing* see also *IEEE Transactions on Signal Processing* 32(2), 304–337.

de Lathauwer L, de Moor B and Vandewalle J (2000) Fetal electrocardiogram extraction by blind source subspace-separation. *IEEE Transactions on Biomedical Engineering* 47(5), 567–572.

Döhler R (1991) Squared Givens rotation. *IMA Journal of Numerical Analysis* 11(1), 1.

Eleftheriou E and Falconer D (1986) Tracking properties and steady-state performance of RLS adaptive filter algorithms. *Acoustics, Speech, and Signal Processing*; see also *IEEE Transactions on Signal Processing* 34(5), 1097–1110.

Eweda E (1998) Maximum and minimum tracking performances of adaptive filtering algorithms over target weight cross correlations. *Circuits and Systems II: Analog and Digital Signal Processing*, see also *IEEE Transactions on Circuits and Systems* 45(1), 123–132.

Eweda E and Macchi O (1987) Convergence of the RLS and LMS adaptive filters. *IEEE Transactions on Circuits and Systems*, 34(7), 799–803.

Gentleman W and Kung H (1981) Matrix triangularization by systolic arrays. *Proc. SPIE* 298, 19–26.

Givens W (1958) Computation of plane unitary rotations transforming a general matrix to triangular form. *Journal of the Society for Industrial and Applied Mathematics* 6(1), 26–50.

Hamill R (1995) VLSI algorithms and architectures for DSP arithmetic computations, PhD Thesis, Queen's University Belfast.

Haykin S (2002) *Adaptive Filter Theory*. Beijing: Publishing House of Electronics Industry, pp. 320–331.

Hsieh S, Liu K and Yao K (1993) A unified approach for QRD-Based recursive least-squares estimation without square roots. *IEEE Transaction on Signal Processing* 41(3), 1405–1409.

Hudson J (1981) *Adaptive Array Principles*. Institution of Engineering and Technology.

Kalouptsidis N and Theodoridis S (1993) *Adaptive System Identification and Signal Processing algorithms*. Prentice-Hall, Upper Saddle River, NJ, USA.

Lightbody G (1999) High performance VLSI architectures for recursive least squares adaptive filtering, PhD Thesis Queen's University Belfast.

Lightbody G, Woods R and Francey J (2007) Soft IP core implementation of recursive least squares filter using only multiplicative and additive operators *Proc. Int. Conf. on Field Programmable Logic*, pp. 597–600.

Lightbody G, Woods R and Walke R (2003) Design of a parameterizable silicon intellectual property core for QR-based RLS filtering. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, **11**(4), 659–678.

Liu K, Hsieh S and Yao K (1990) Recursive LS filtering using block Householder transformations. *Proc. Int. Acoustics, Speech, Signal Processing Conf.*, pp. 1631–1634.

Liu K, Hsieh S and Yao K (1992) Systolic block Householder transformation for RLS algorithm with two-level

pipelined implementation. *IEEE Transactions on Signal Processing* 40(4), 946–958.

McCanny J, Ridge D, Hu Y and Hunter J (1997) Hierarchical VHDL Libraries for DSP ASIC Design. *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Proc.*, p. 675.

McWhirter J (1983) Recursive least-squares minimization using a systolic array. *Proc. SPIE* 431, 105–109.

Morgan D and Kratzer S (1996) On a class of computationally efficient, rapidly converging, generalized NLMS algorithms. *Signal Processing Letters, IEEE* 3(8), 245–247.

Rader C (1992) MUSE-a systolic array for adaptive nulling with 64 degrees of freedom, using Givens transformations and wafer scale integration. *Proc. Int. Conf. on Application Specific Array Processors*, pp. 277–291.

Rader C (1996) VLSI systolic arrays for adaptive nulling. *Signal Processing Magazine, IEEE* 13(4), 29–49.

Rader C and Steinhardt A (1986) Hyperbolic householder transformations. *IEEE Transactions on Signal Processing* 34(6), 1589–1602.

Shan T and Kailath T (1985) Adaptive beamforming for coherent signals and interference. *IEEE Transactions on Signal Processing* 33(3), 527–536.

Shepherd T and McWhirter J (1993) Systolic adaptive beamforming. *Array Signal Processing* pp. 153–243.

Tamer O and Ozkurt A (2007) Folded systolic array based MVDR beamformer. *ISSPA 2007, International Symposium on Signal Processing and its Applications, Sharjah United Arab Emirates*.

University N (2007) Variable precision floating point modules. Web publication downloadable from http://www.ece.neu.edu/groups/rpl/projects/floatingpoint/index.html.

Walke R (1997) High sample rate givens rotations for recursive least squares, PhD Thesis, University of Warwick.

Wiltgen T (2007) Adaptive Beamforming using ICA for Target Identification in Noisy Environments. MSc Thesis, Virginia Polytechnic Institute and State University.

第 13 章　低功率 FPGA 的实现

13.1　引言

正如本书引言中所表明的，研究人员做了许多工作来提高技术水平，并且也研究了如何增加晶体管的数量及提高单个晶体管的运行速度。在 SoC 设备平台出现期间，FPGA 技术结构上的变化已经成为 FPGA 技术成长的核心内容。这个技术的改革极大地推进了 FPGA 技术的成功，因为可编程性能使得使用者能开发出高度的并行化及相当适合 DSP 应用的流水线式的电路架构。这种控制程度可以让使用者开发出相当高的性能，而且这并不需要依靠像在处理器实现上的较高的时钟频率。

功耗的问题需要特别注意。随着科技的进步，功耗也相应下降，有人就认为功耗应该不再是一个问题。虽然这个问题是与单个实现有关的晶体管数量的增加相关，因此提高了单个芯片功耗。可是这仅仅是问题的一方面。主要的问题是在硅芯片中功耗的本质也随着科技进步变化着，而且这对 FPGA 也有着重要的影响。这就出现了一种趋势，那就是多年前被当作微不足道而且几乎被忽略的泄露功率现在开始起支配作用，这对所有微电子设备来说都是一个问题。然而，需要对芯片内的连接体充放电的功率与在前十年逐渐成为重要问题的晶体管需要的功率相比，由于提供的可编程互连而与 FPGA 的实现紧密相关。这称作动态功耗，而且它会随着科技进步越发恶劣。因此意味着当与 ASIC 技术中的同等设计相比，在 FPGA 的实现中连接体重要得多。当然，FPGA 的可编程本质意味着我们能够做些什么来减小功耗，这对处理器来说就不是这么回事了，其中为了减轻可编程性而开发的底层构架是十分不适合低功率实现的。

对于减少功耗，有着许多重要的原因。功耗的增加会导致设备温度的提升，如果不控制好就会对设备的可靠性造成很不好的影响。低功耗对降低热量的发散是有影响的，这能使温度管理的成本变得更低，依据封装成本分配芯片中的热量，因此避免了在微处理芯片上出现大量“铝塔”，并且也减少了成本或甚至减少了利用基于风扇让系统降温的需求，这个风扇系统通常需要更好的空气流动来降低整个板子表面的温度。Xilinx Inc. （2007）中说到“设备工作温度降低10℃可以转变为两倍的元件寿命”。一个低功率的 FPGA 实现对整个系统的供电设计有最直接的收益，这使得使用的元件数量更少、价格更低。比如，一个高性

能能量系统的实现成本估计在每瓦特 0.5～1.0 美元（Xilinx Inc. 2007）。因此，减少 FPGA 的功耗在成本和可靠性上都会有明显的影响。

因为在没有提高系统成本或系统规模的前提下，电池技术并没有为能量的提高提供任何的创新，使得便捷性更加依赖于设备功耗的降低。随着新的服务引入移动终端硬件，系统性能的预算也会随之提高，而这一定是在目前减小了的功率预算中实现的。为了降低危险，就意味着需要使用可编程的硬件来解决，而这就直接意味着较高的功耗。

本章的目的是概述功耗源头及能够减少功耗源头的技术。从 13.2 节开始着眼于不同的能源及通过半导体技术蓝图（IRTS 2003）来预测它们将来的变化。13.3 节将简洁回顾减少功耗的可能方法。将在 13.4 节介绍电压的降低，它是减少静态功耗的一种技术。但是，通过 FPGA 中的电压缩放来减少功率是有限制的，这都是为了安全地操作设备，需要小心地选择电压，在 13.5 节中将会有大量的减少动态功耗技术的概述。其中包括流水线的使用、局部的强制性、数据重排和固定参数的开发利用。在 13.10 节中，这些技术的一部分会被应用到 FFT 和实际功率的设计上。

13.2 能源消耗

CMOS 技术中的功耗会被分为两部分，也就是静态功耗和动态功耗，静态功耗正如其名是当电路在以静态模式开启并处理必要数据时消耗的功率，而动态功耗则是芯片运作时的功耗。静态功耗对备用模式下的电池寿命很重要，因为它表示了上电时的功耗。动态功率对运行时的电池寿命有重要作用，因为它代表了在处理数据时的功耗。静态功耗包含了如图 13-1 所示的大量组成部分。栅极漏电是栅极到基极的电流，源极到漏极（阈值电流）漏电即认为设备是关闭的时候在通道中从漏极流向源极的电流（也就是栅极与源极之间的电压 V_{GS} 比晶体管的阈值电压 V_T 要低），以及当源极和漏极的电势高于基极时，通过晶体管的源极到基极的结和漏极到基极的结的反向偏置 BTBT 电流。

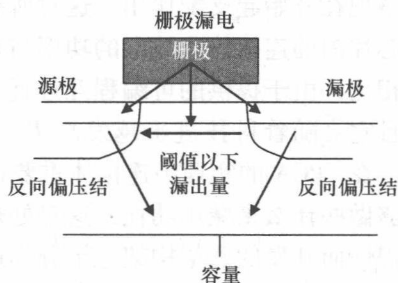


图 13-1　在 MOS 晶体管中泄露源部分

13.2.1 动态功耗

对于动态功耗，通过如图 13-2 所示的一个简单反相器的泄露量将其纳入了考量。假设一个脉冲的数据馈入到对设备充放电的晶体管中。当栅极使得它们的

输出改变为一个新变量，并且栅极是依靠 CMOS 反相器中晶体管 p 极 n 极的电阻值时就会消耗功率。

因此，通过电容电流和其电压可以表示为式（13-1）（Wolf 2004）。

$$i_{\mathrm{C}}(t)=\frac{V_{\mathrm{DD}}-V_{\mathrm{SS}}}{R_{\mathrm{P}}}\mathrm{e}^{(-t/R_{\mathrm{P}}C_{\mathrm{L}})} \tag{13-1}$$

在电容充电时的功率要求如式（13-2）。

$$v_{\mathrm{C}}(t)=V_{\mathrm{DD}}-V_{\mathrm{SS}}[1-\mathrm{e}^{(-t/R_{\mathrm{P}}C_{\mathrm{L}})}] \tag{13-2}$$

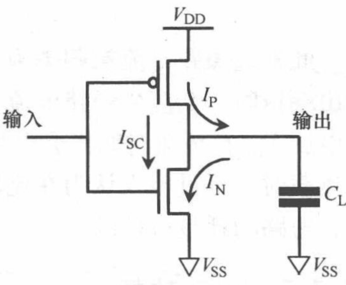


图 13-2　简单 COMS 反相器

给电容器充电所需要的能量为

$$E_{\mathrm{C}}=\int[i_{C_{\mathrm{L}}}(t)V_{C_{\mathrm{L}}}(t)]\mathrm{d}t \tag{13-3}$$

$$=\left[C_{\mathrm{L}}(V_{\mathrm{DD}}-V_{\mathrm{SS}})^2\left(\mathrm{e}^{-t/R_{\mathrm{P}}C_{\mathrm{L}}}-\frac{1}{2}\mathrm{e}^{-2t/R_{\mathrm{P}}C_{\mathrm{L}}}\right)\right]_0^{\infty} \tag{13-4}$$

$$=\frac{1}{2}C_{\mathrm{L}}(V_{\mathrm{DD}}-V_{\mathrm{SS}})^2 \tag{13-5}$$

当电容放电时，同样量的电荷分散并穿过 n 型晶体管，在晶体管运行的一个周期里，电容消耗的总能量为$\frac{1}{2}C_{\mathrm{L}}(V_{\mathrm{DD}}-V_{\mathrm{SS}})^2$。若这被纳入设计的正常运行中，其中假设在时钟频率 f 下同步运行，那么就能得出总功耗为$\frac{1}{2}C_{\mathrm{L}}(V_{\mathrm{DD}}-V_{\mathrm{SS}})^2f$。但是，这就假设了每个晶体管都以时钟频率进行充放电（这是不可能发生的）。因此引入 α 来指示有多少晶体管在充电。对于不同的电路，α 的值是不同的，见表 13-1（Chandrakasan 和 Brodersen 1996）。

表 13-1　典型转化率（Chandrakasan 和 Brodersen 1996）

信号类型	转换率（α）
时钟信号	0.5
随机数据信号	0.5
随机数据驱动的简单逻辑电路	0.4 ~ 0.5
有限状态机	0.08 ~ 0.18
视频信号	0.1（MSB）~ 0.5（LSB）
结论	0.05 ~ 0.5

$$P_{\mathrm{dyn}}=\frac{1}{2}C_{\mathrm{L}}(V_{\mathrm{DD}}-V_{\mathrm{SS}})^2f\alpha \tag{13-6}$$

当 V_{SS}假设为 0 时，式（13-6）可简化为式（13-7）。

$$P_{\mathrm{dyn}}=\frac{1}{2}C_{\mathrm{L}}V_{\mathrm{DD}}^{2}f\alpha \tag{13-7}$$

此外，短路电流被归类为动态功耗。当栅极的输入信号上升/下降时间大于输出端口时，会产生短路电流，这就产生了输入和输出间的不平衡，这也意味着供电压 V_{DD}在极短时间内会发生短路。这特别会发生在当晶体管驱动一个大的容性负载时，不过有人认为在比较好的设计中，这是能够避免的。因此在某种程度上，短路功耗是可控的。

13.2.2 静态功耗

缩放技术为大量的产品革新提供了动力，因为它能缩小晶体管的尺寸，如图 13-3 所示。简单来说，通过 k 来缩小晶体管意味着图 13-3b 所示晶体管的新尺寸是 $L'=1/k(L)$，$W'=1/k(W)$且 $t_{\mathrm{ox}}=1/k(t_{\mathrm{ox}})$。显然，晶体管的数量会增加 k^2 且晶体管的速度也得到提高；随着晶体管中电流的减少，电流晶体管中的功耗也会按预期降低。

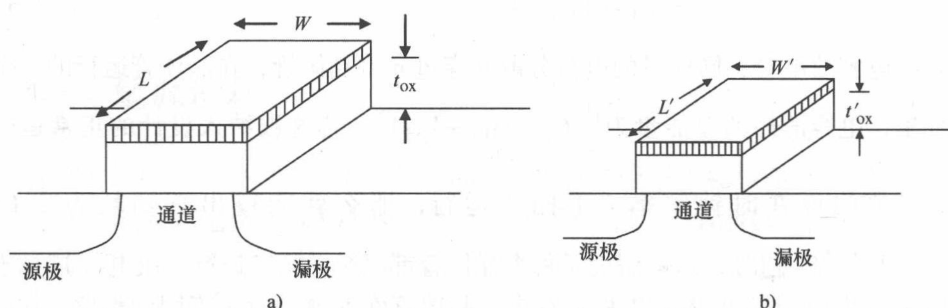


图 13-3 晶体管缩放的影响
a）原始工艺 b）复合源

然而功耗的预期降低并没有发生。为了避免缩小后的结构中电场过大，有必要减小供电电压 V_{DD}。反过来阈值电压 V_{th}也需要减小，否则晶体管将不能正确关闭。阈值电压的减小导致阈值电流的增大。为了处理因为缩小晶体管尺寸而导致的通道变短的影响，所以需要减小氧化层厚度，这也导致了要通过晶体管的栅极绝缘层稍加困难，从而产生栅极漏电。因此栅极泄漏量与栅极氧化层成反比，而这个氧化层为了性能的提高会继续减小，这就导致问题越发恶化。

缩小设备也需要在源极到基极和漏极到基极结点的附近使用较高的掺杂浓度，这都是为了减小耗尽区。但是在较高的反偏电压下，这会产生穿过这些结的极大的 BTBT 电流（Roy 等 2003）。其结果就是缩放导致了这些组件的泄露量急剧增加，并且随着结点温度的升高，因为漏电影响的增加，导致情况越加恶化

（Xilinx Inc. 2007）。

随着晶体管数量的增加，主要的问题是它们对静态功耗的影响在增加。这是从图 13-4 中得到的概述，这张图摘自国际半导体技术蓝图（IRTS 2003），（Kim 等 2003），这个图中的交叉点出现在 90nm 及更小的技术结点处，这一点对大量应用来说是静态功率开始超越动态功率。

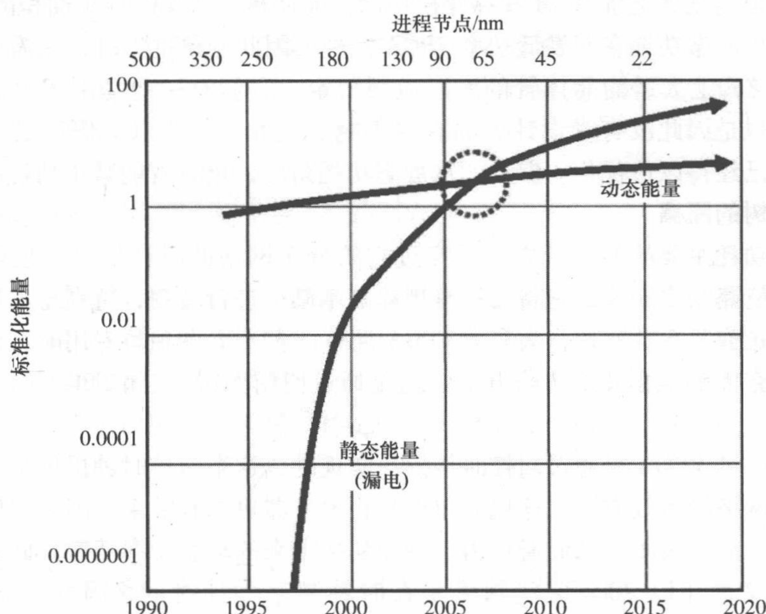


图 13-4　随着技术革新静态能量功耗相对动态能量功耗的影响（Kim 等　2003）摘自© IEEE

这张图对大量的技术都有着很大的影响，正如它现在表明的，基本没有影响设备动态处理的前十年的功耗问题，使得设计者能减小它的影响，在设备的正常待机运行方式上将预测不到功耗。这将对固定的系统设计有影响，但这主要是对无线应用。之后将会采用许多方法来处理。

通过在 90nm FPGA 中使用三栅极氧化层（Xilinx Inc. 2007），Xilinx 曾经试图在 Virtex－4 及接下来的设备中处理庞大的静态功率的影响。使用三栅极氧化层来表示在 FPGA 中使用的三层氧化层。薄的氧化层是给 FPGA 核中的小的、快速的晶体管使用的，厚的氧化层是用于 I/O 接口的更高电压的摇摆晶体管，它不需要很快的速度，而一个称为中间厚度氧化层的三层氧化层则用在配置处理器元件及互连传输晶体管中。这些晶体管运行的阈值电压要高于薄氧化层的晶体管，但是因为它们仅仅用于为逻辑和数据传输来存储配置，所以对速度没有严格的要

求。在减小功耗上这无疑是一个明智的取舍，而且它并不适用于除 FPGA 以外的技术，比如微处理器和 DSP 处理器，除非它们有存储配置数据的晶体管。

Altera 使用了相似的策略，即使用两种类型的晶体管，第一种是有着低阈值电压及较小最小通道长度的晶体管，这种晶体管是为了在 DSP 块和逻辑元件中高速地运行；第二种是有着更高阈值电压及较大通道长度的一种低功耗晶体管，这是为了根据比如配置 RAM 和存储模块的性能使得芯片实现更小面积的要求。

当处理静态功耗在开发低功率 FPGA 实现上因此变得重要时，它需要视使用在 FPGA 结构上大量的晶体管的底层应用而定；它们有一些是用于存储和控制的，而且也是因此没有受设计者的直接影响。此外，为了以上概述的低功率性能，设备已经得以最优化。但是，从静态功耗角度，仍有着能减小功耗的技术。

时钟树的隔离

静态功耗主要的参与者之一是穿过它的分布网络的时钟信号，也称作时钟树，而且电路与之相连，它将会在有规律的基础上进行转变，特别是如果设计为同步的。正如第 5 章所述，大多数 FPGA 拥有许多 PPL 和独特专用时钟树网络时钟信号，它能够根据要求进行开关。这是通过使用 MUX 关闭 FPGA 的一部分来实现的。

Virtex－4 FPGA 是通过门控时钟模块实现的，这个门控时钟提供了一个关闭全局时钟网络的有效方法。这比简单地禁止触发器更加有效果，因为它使得较大的切换网络能够关闭，从而避免由于剧烈切换带来的动态功率消散。此外，应该注意到不像式（13-7），这些网络是在时钟频率下计时的，因此功率会减少许多。

13.3 降低功耗的技术

通过检验式（13-4），很明显对于动态能量功耗有许多因素都是可以改变的。供电电压 V_{DD} 已经被 FPGA 供应商预先确定了（并且最佳化地提供低功耗处理），并且使用者能通过设计软件有效利用这个电压减小的范围。这就只剩下了其他参数的调整，也就是时钟频率 f、转换速率 α 及负载电容 C_L 决定的触发率。但是调整时钟频率 f 和转换率 α 的任何技术应该在理解到一点的前提下得到开发，这一点就是系统的整个时钟速率一般由应用决定，以及转换率将由应用领域进一步管理，这意味着表 13-1 应该得到重视。

一般来说，减小功率的技术要么是最小化转换电容 Cf，要么使用通过提高系统吞吐量来减小供电压，要么通过并行硬件的使用从而增加面积和电容量或提高速度（Chandrakasan 和 Brodersen 1996）。电压的减小会让性能慢下来直到在较低的功耗预算下达到要求的吞吐率。后一种方法使得功率成二次方减小，但这是

以区域的线性增长为代价的，也就是 C_L 和/或频率 f。在 FPGA 中这是更加困难一点，但是在 Chow 等（2005）中有合适的解决技术，这将在第 13.4 节中介绍。

电容量和转换率还有减小的余地，但不能将它们分开考虑，思考减小电路的开关电容是很有用的，也就是每个节点的所有转换率乘上该节点电容量的总和。与电路电容截然相反，这是功耗上一个很重要的措施，因为一个电路要么是不会有太大功耗且有着“低”转换率的“较大”电容网络，要么是有着较大功耗且有相当“大”转换率的大量“低”电容量网络。在转换率层上应用有同样的争论；一些网络有着较高的转换率，但是电容量却很低。许多技术也都进入了这个领域，因此在第 13.5 节中有详细的阐述。

13.4　FPGA 中电压按比例缩小

正如名称所示，电压按比例缩小涉及要在保证电路正常工作的前提下减小电路的供电电压。比较有代表性的是，设计者将通过设计技术开发一切的电压容量来放慢电路的运行，可能通过减小一定的面积或一些其他的增益来实现。降低电压也许会造成电路故障，像关键路径定时也许就会产生故障。这是因为缩小电压会产生电路时延 t_d，正如式（13-8）所确定的（Bowman 等　1999）。

$$t_d = \frac{kV_{DD}}{(V_{DD} - V_t)^2} \tag{13-8}$$

式中，k 和 α 为常数，且 $1 < \alpha < 2$；当 V_{DD} 缩小时，电路时延 t_d 也会增加。

电压缩小的影响能通过两方面来处理，当电压缩放发生在一个具体的 FPGA 实现中时，增加电路来准确监测并且监测出正确的电压阈值来实现低功率运行（Chow 等　2005），另一方面是应用设计技术来提高电路的运行速度。

其中的一种方法是提高运算速度，如图 13-5 所示。如果假设如图 13-5a 所示操作在运行电压下匹配速度要求，然后要么将并行添加到电路实现中，要么对并行进行开发（在许多 DSP 系统中通常是这样的），则能产生如图 13-5b 所示电路。现在这个电路运行得比原始电路更加快，因此电压缩小的应用是有可能的；又由于 V_{DD}^2 电压的缩小和频率的缩小会导致功率的降低。当然，图 13-5b 所示电路会有一个更大的面积、电容量及转换率，但是二次方式的，以及频率的减小会得不偿失。

考虑电压缩小能应用于 FPGA 的可能方法是很重要的。这应该仅仅应用于 FPGA 核，因为 FPGA 核的应用范围更加宽广；这是可能的因为已经为核和 I/O 接口准备了大量的 V_{DD} 引脚。I/O 接口按照已经设计出的规范来运行是很重要的，特别像一些静态功耗技术将会被应用到 FPGA 设备上，比如 Xilinx Virtex-4 和-5 系列及 Alter’s Stratix Ⅱ和Ⅲ FPGA 系列。降低电压会增加电路的时延，但

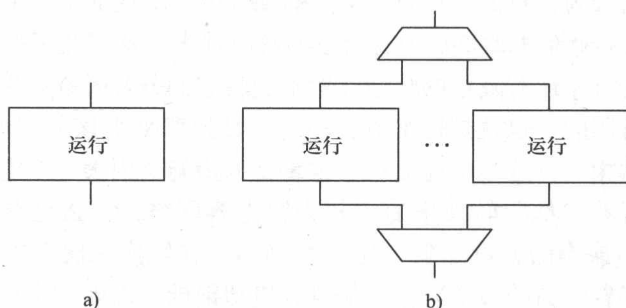


图 13-5 使用并行（和电压的缩放）来降低能量功耗
a）原始实践 b）并行实践

是考虑到在最小的电路时延下仅仅只有部分电路需要运行，因此在没有影响性能的情况下减小大部分电路的电压是有余地的。当然，要通过一系列设备的设计必须可靠并且在时延和运行温度中可以有变化。

在 Chow 等（2005）中遵守假设原始电路、改变内电压供应然后检查任何可能错误的第一原则（虽然在图 13-5 中被理所当然地用做第二种方式）。作为在正常运行时电压缩放的结果，设计师能观察出任何两种设计错误，他们认为在这个基础上，这只是找出其他两类错误的一个例子：一种是 I/O 错误，那是在当低电压的核电路必须连接在原电压下运行的 I/O 上时产生的；另一种是时延错误，那是关键路径没有满足定时要求时发生的。在 I/O 错误中，其中的危机是来自核心的高电平输出信号对于 I/O 缓冲的阈值电压来说太小，以至于不能够被正确地侦测到；这意味它会被不正确地译为一个低值。

Chow 等（2005）中使用一个逻辑时延检测电路（Logic Delay Measurement Circuit，LDMC）（Gonzalez 等 1997），它除了被应用于设计者的 FPGA 电路上，它还由一条时延线或反相器链、一连串的触发器和通过用正规的 FPGA 逻辑资源实现的先行零检测器组成。这个电路通过向前传递穿过反相器链的一个波形来有效地产生一个警报信号，然后测量出通过使用同样时钟信号有多少触发器能正确为数据计时。因此已经转变过的输入的数量会依靠反相器的时延，而反相器的时延反过来会依靠温度和供电电压；通过使用这种先行零检测器，能够计算出电路的传输时延。用这种方法，LDMC 就可以测量出在半个时钟周期中下降沿传递的时延数。对这个电路的研究能使许多电路受益，而且据作者说明，能量减少量在 4% ~54% 的范围内，然而他们期望实现令人满意的 20% ~30%。这种方法主要的限制就是由于 FPGA 之间电路性能不同，因此必须在具体环境里研究每个 FP-GA 的运行。考虑到功耗、时延和温度之间的关系，不能假设设备之间的性能相同。通过在最坏情况下运行的规范化，这在正常的 FPGA 运行中被避免了。但

是，它们是许多的其他低功率设计技术，这些技术不要求改变 FPGA 参数，不过这些参数也许是在采用这样一个方法之前，设计者想要研究的。

13.5　开关电容的减少

之前的技术要求电压被缩小（主要的仅仅是内电压），但是没有对这种缩小的应用结果进行处理。但是，正如第 13.3 节介绍的，另外一种减小功耗的技术涉及减少电路的开关电容。如式（13-4）所示，这使得动态功耗二次方式地减小（$f \times C_L$ 结果）。最后的 FPGA 布局能很大程度影响开关电容，因为布局决定的第一点就是路由互连的电容，而且因此在一定程度上决定转换率，这都是由于各个路径上的不同时延（虽然所有的开关会通过输入数据和电路功能进行管理）。但是架构风格会在很大程度上影响 FPGA 的布局，正如会在本节及 FFT 例子中（第 13.10 节）将看到的，这意味着架构的确定会对功耗产生很大影响。

一些技术会得到考虑，在文献中这些都能很好地得到理解。这些技术囊括的并不是很详细，而且这也代表了仅仅是一些合适的最优化才有可能实现。

13.6　数据的重新排序

在第 4 章介绍的 DSP 处理器的实现中，构架主要包括数据和经数据总线连接到处理器的程序储存器。因此在这些架构中，总线的电容会被固定，但是在一些情况下为了最小化汉明距离可能会重排数据计算，从而在大的电容总线上实现转换率的减小。看一看下面列出的有着 4 位参数的 4 抽头滤波器。

$$
\begin{array}{llc|llc}
a_0 & 1011 & & a_0 & 1011 & \\
 & & 3 & & & 1 \\
a_1 & 0110 & & a_2 & 1001 & \\
 & & 4 & & & 3 \\
a_2 & 1001 & & a_3 & 0100 & \\
 & & 3 & & & 1 \\
a_3 & 0100 & & a_1 & 0110 & \\
 & & 4 & & & 3 \\
a_0 & 1011 & & a_0 & 1011 &
\end{array}
$$

8　次转换 3　次转换

能够看出如果将参数加载到正常的序号中，也就是 a_0，a_1，a_2，a_3 然后回到 a_0，这需要转换 14 次。这涉及通过连接体线性电容的充放电来将这些参数加载入处理引擎中。根据确定出的架构，这个互连长度还是相当可观的。比如，选择一个 μ 型 DSP 的实现，其中的数据是经较长的总线从参数存储器加载进来的，因此这能极大地促进开关电容的使用。通过改变加载的序数，转变的次数能够减小到 8 次。然后主要的问题是解决参数的无序运行。同时在一些电路架构的实现

中这会是一个问题，在这些架构中，参数的选择是通过固定的数据选择器来完成的，这是不能被改变的，并且这对 DSP 的实现而言并不是那么回事。这在参数是预先确定并且不会改变的应用中运作得很好。如果情况属实，那么通过使用大量在第 8 章中概述的技术来开发电路架构，从而产生一个使用这项技术的架构就是可能的。比如软件共享能够基于参数的交换率来进行控制。

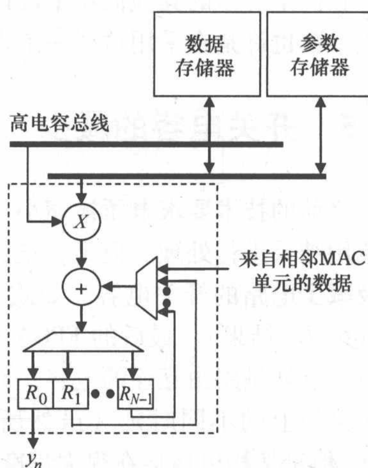


图 13-6 通用 MAC 时域滤波器的实现（Erdogan 和 Arslan 2002）

在 Erdogan 和 Arslan（2002）中，作者展示了应该如何将设计应用于 FIR 滤波器中来使得功耗减少 62%。图 13-6 所示的这个架构便是从中引入且表示了一个包含乘法器和加法器的 MAC 结构，而它们需要依靠存储器中的程序和参数数据来运行。通过将前面部分的输出注入到电流块，能够将提出的这个架构进行级联。这个结构显然支持无序运行，这就导致经参数数据总线传输的来自参数存储器的交换数据的减小。通过开发直接型而不是转换型 FIR 滤波器的实现，同样减少了当一个数据字被装载然后重复利用时在数据总线上的交换。

13.7 固定参数的运算

第 6 章提出的 DA 和 RCM 技术是一种最优化技术，这种优化能被应用在除了重排序数的技术中。最初提出这些技术是因为它们能够减小面积并且在某些情况下提升速度。但是，这两种方法的根本观点是减少需要用于执行一个完整乘法计算的硬件数量，而且也因此隐晦地减少了需要构造乘法器和相应开关电容互连的数量。至于 RCM，只有乘法器结构的一部分用于计算，因此有着构造额外多路复用的空间，通过它便能够关闭一部分的乘法器，虽然这还没有得到详细的研究。

13.8 流水线

降低功耗的一个有效方法是将流水线与具有功耗意识的布置元件相结合。流水线如图 13-7 所示，将图 13-7a 所示原电路的过程分成如图 13-7b 所示的几个简短的过程，从而提升速度，但是根据时钟周期而非实际时间，时延会增加

（因为时钟周期已经变短了）。处理速度上的提升能以一种相似的方式用在图13-5并行方式的使用上，从而减小电压，实现功率的降低（Chandrakasan 和 Brodersen 1996）。的确如之前章节所述，两种技术都能应用于提供相当可观的速度提升。

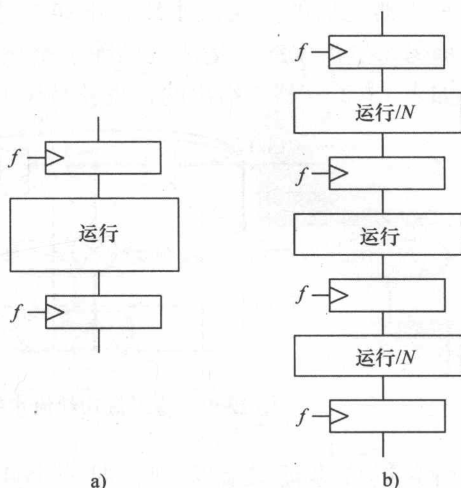


图13-7　流水线的应用
a）原本的实现方式　b）流水线的实现方式

除了提高速度外，流水线还可提供一个非常有用的机制来降低功耗（Raghunathan 1999，Keane 等 1999），这在FPGA中更明显（Wilton 等 2004）。在FPGA设计中，空间和路径工具的目的是达到最好性能的实现，从而达到所需速率（和更低的低功耗实现）。流水线为设计提供了更严谨的结构，从而使设计可达到更快的定位并减少较长的路径数量。此外，流水线电路比非流水线电路的故障更少，这是因为它通常在寄存器之间有更少的逻辑层，这会使动态功耗更少。

这种技术对FPGA特别有效，由于较长的路由路径和可编程的开关，因此使得耗费一定的功率预算就能提高其灵活性。这些特点提供了可编程性，但是却充满了寄生电容（Chen 等　1997），如图13-8所示，其呈现出一种典型的FPGA路由的一种模式。在一个FPGA中实现流水线的另一个好处就是它也许能够使用未充分利用的资源，也就是在逻辑元件输出上的触发器，从而提供一定面积的增长（Wilton 等2004）。同时有人提出触发器不会使用在非流水线式的类型上，而且因此对功率没有影响，应该注意的是仍然可能会有时钟提供给它，这会使之主导着功耗。

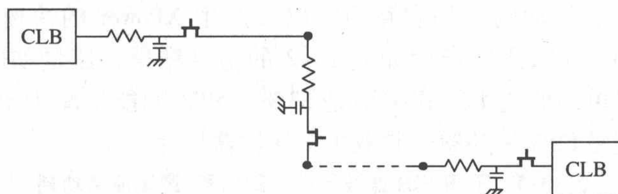


图13-8　典型FPGA互联路由

看一看图13-9中给出的对FIR滤波器的流水线式的应用。这个滤波器是一个如之前章节中介绍的简单时延线滤波器结构。这将对两重流水线进行研究，也

就是乘法器之后的一层流水线，正如通过 Cut2 表明的和表格 13-2 中 PL1 给出的，而另一层流水线则是插入到除了第一层流水线的加法器（和时延）链 Cut1 中，如表 13-2 中 PL2 给出的，也就是说 PL2 包括了 Cut1 和 Cut2.

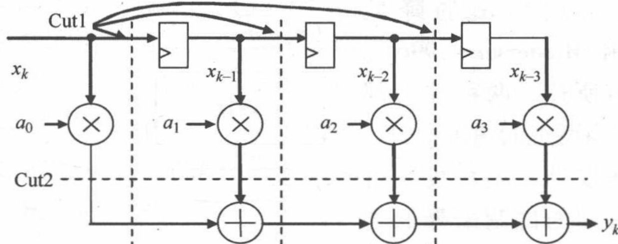


图 13-9 为了低功耗流水线式 FIR 滤波器的实现

研究过许多滤波器实现，即一个 4，8，16 和 32 抽头的 FIR 滤波器在 Vertex-Ⅱ XC-2V30000bf 957-6 FPGA 得到了实现（McKeown 等 2006）。这个滤波器会通过使用地址总线、数据总线和使能信号来加载参数，以此初始化滤波器，这也是为了使之与所有的实现保持一致。使用无截断，并且输出字长度对应 4 抽头和 32 抽头的滤波器分别是 24 位和 27 位。为每个滤波器尺寸定义一个字增长变量，通过这个变量来进行扩展，以此避免截断。在应用降低技术之前，非流水线式的类型（PL0）被用于测试功耗。除了通过流水线站强加的延时外，在所有类型中的功能都是一样的。使用一个语音数据文件来模拟时钟频率为 20MHz 和仿真时间为 200μs 的滤波器设计。

FIR 滤波器是以 VHDL 语言编码的，将通用性参数用作系数和输入数据字的尺寸。在 Matlab® 中创建 4，8，16 和 32 抽头的滤波器，以及如输入般提取和使用滤波器参数而构建了三种滤波器（带通、低通和高通）。ModelSim™ 和 Synplicity® Synplify Pro® 分别是用来仿真和综合的。Xilinx ISE™ Project Navigator（6.2 版本）用于转换、绘制映射、布局和路由，而且ISE™设计的一套子程序已用于编译元件库、手动的布局和路由，以及产生 XPower 的快速布局和路由的 VHDL 文件。Xpower 被用于产生如表 13-2 的仿真结果，流量如图 13-10 所示。从结果可得知对单站的流水线式类型是 63% ~59% 的能量减少量，并且对双站的流水线式类型是 82% ~98%，其取决于滤波器尺寸。

表 13-2 不同 FIR 滤波器的内部信号/逻辑能量功耗

方法	滤波器抽头尺寸			
	4	8	16	32
PL0	8.4	89.7	272.0	964.2
PL1	3.1（-63%）	29.1（-68%）	89.7（-67%）	391.7（-59.3%）
PL2	1.5（-82%）	6.7（-93%）	8.1（-97%）	16.4（-98%）

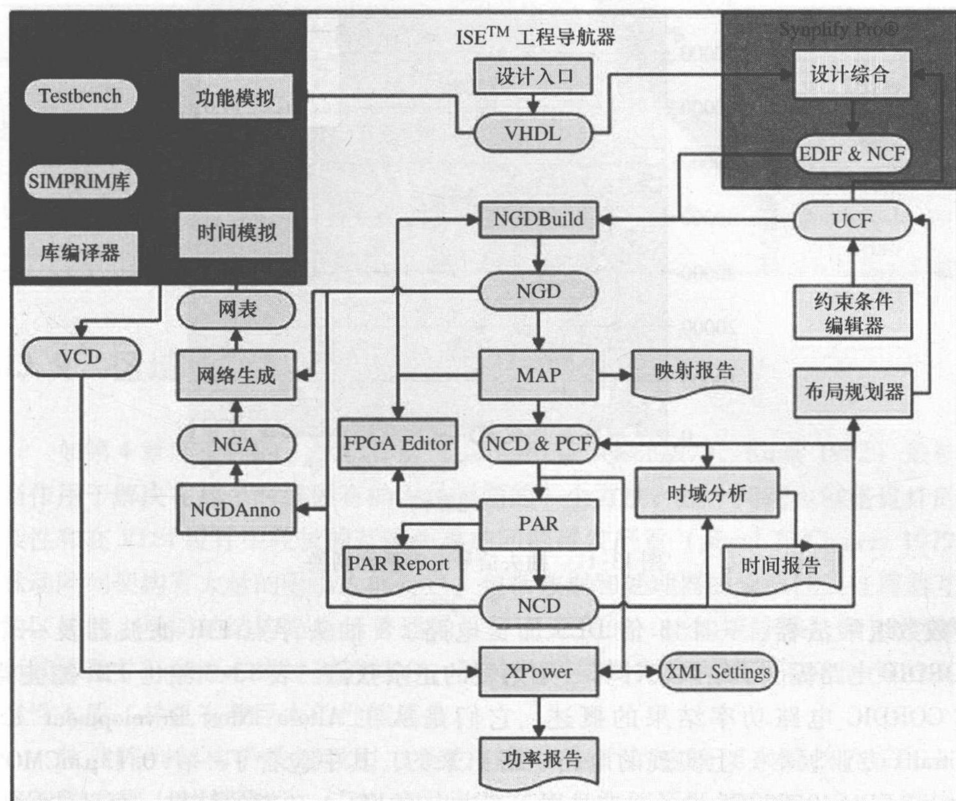


图 13-10 Xpower 近似的详细设计流程

从表格中可以看出随着滤波器的变化，很明显流水线变的更加有效率，而且因此设计的面积和互连的长度都在提高；这就为具有功耗意识的布局提供了更好的机会。通过紧密布局互相连接的原件，网络长度也变短了。这从两方面减少了功耗，即通过减小网络电容，以及通过减小不规则的传输时延。将设计分成多个流水线站，通过减小传播时延的连锁影响来进一步减小功耗。如图 13-11 所示，它也表明了快速布局及路由电容量都四舍五入到最近的整数，它是依靠有着等效电容网络上的开关率的综合绘制出来的。绘制了 PL0，PL1 和 PL2 类型的这些值，而且也显示出不仅仅是在高电容网络中降低的转换率，而且当功率降低技术实现时，设计中的开关数会更少。

在 Wilton 等（2004）中提出的结果是更加彻底的，因为他们是基于来自真正面板装置的数据，而且它们同样让人印象深刻。他们已经为一个 64 位无符号

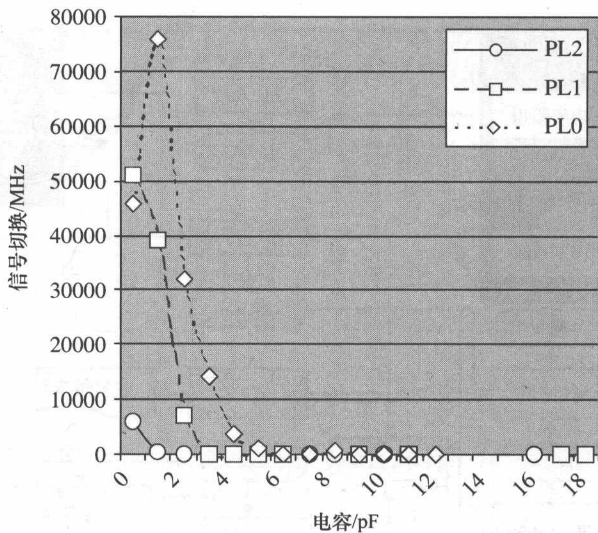


图 13-11 抽头信号电容量和切换

整数数组乘法器、一个 3 倍 DES 加密电路、8 抽头浮点 FIR 滤波器及一个 CORDIC 电路提供了结果来计算一个角度的正余弦值。表 13-3 给出 FIR 滤波器和 CORDIC 电路功率结果的概述。它们是从在 Altera Nios Development Kit（Stratix 专业版本）上实现的电路中得出来的，其中包含了一个 0.13μmCMOS Stratix EP1S40F780C5 设备。这是用于产生出原 FPGA 功率的结果，而且估计功率来自 Quartus 模拟器和功率估计器。

这个结论显示了不同重流水线应用的影响。他们引用了所有 40% ~82% 的储存量并表明当从结果中提炼出静态能量时，在动态逻辑模块能量上的节省量可以高达 98%。他们表明低级物理优化设计仅能够让能量节省上升 23%，突出了应用系统级最优化的重要性，并且使得流水线式的影响更显著。

表 13-3 对 0.13μm 的 FPGA 的流水线结果

基准电路	流水线站数	估计功率	原 FPGA 功率
8 抽头浮点 FIR 滤波器	2	4420	7866
	4	2468	5580
	最大	776	3834

（续）

基准电路	流水线站数	估计功率	原 FPGA 功率
计算角度正余弦的 Cordic 电路	4	971	5139
	8	611	4437
	16	565	4716
	最大	567	4140

13.9　区域性

如第 4 章所论述的，脉动阵列（Kung 和 Leiserson1979，Kung 1982）最初被当作用于解决在那个时候固有的设计问题的一个方法，这个问题也就是设计的复杂性和在 VLSI 设计中较长的互连距离的问题越发严重（Mead 和 Conway 1979）。脉动阵列架构有大量的吸引人的特点，包括数据和处理器的规律性及处理器互连的区域性。早期的结构是由于普通的计算而开发的，比如矩阵乘法和 LU 分解，而其涉及并行和流水线过程的结合。脉冲阵列架构充分利用了 DSP 计算的高规律性本质，并且有着巨大的性能潜力。

脉动阵列架构的关键吸引力是它们提供一个结构框架来确认在 FPGA 电路架构发展上的规则性。正如前面章节在流水线例子所演示的，一个架构的最佳化能在功耗减少方面提供好处。在 Keane 等（1999）中也清楚地表明如何构建架构，就此而言乘法器的比特层架构能够减小功耗。这个比较是基于 ASIC 的，但是来自那篇文献的关键信息是保护区域性的重要性。图 13-12 所示为进位保留乘法器和 Wallace 树乘法器（见图 3-7）的不同网络长度的转换率（见图 3-6）。

图 13-12a 所示为当规律性在布局过程被保留下来时，转换是如何被限制在一些网络列表长度尺寸上的。图 13-12b 所示为综合期间同样的设计被平坦化时的影响。图 13-12c 所示为 Wallace 树的分布规律。应该注意的是高转换率会发生在更大的网络上。两种设计都有相似的晶体管数量和模拟运行的转换率，并且明显地表明这个影响是因为在实现 Wallace 树中功率提升了 40%。字长越长问题越糟糕。对 FPGA 实现区域的应用在接下来的 FFT 实现方法的应用中将得到演示。

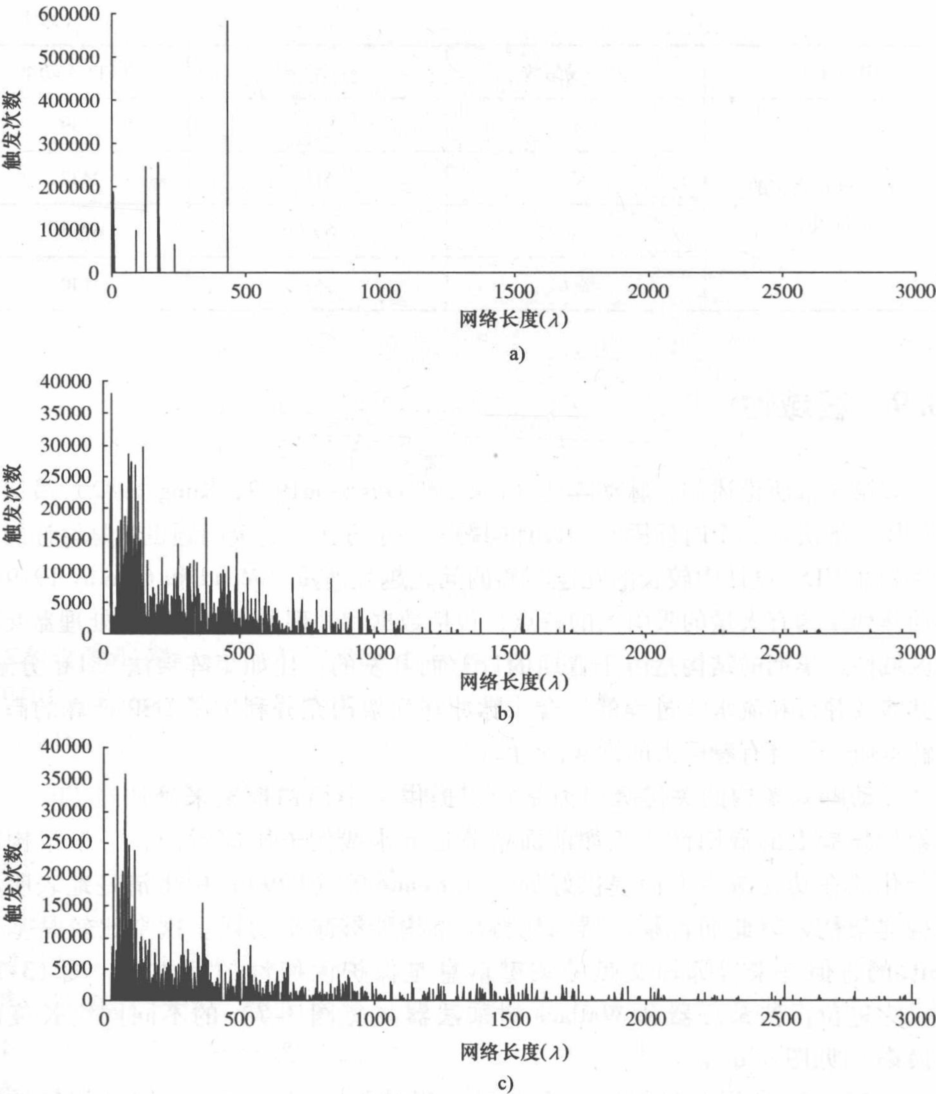


图 13-12　不同乘法器结构的开关电容率

a）保存有规律性的进位的保留　b）基于平面的网络列表的保留进位的乘法器

c）基于平面网络列表的 Wallace 树乘法器

13.10　FFT 实现的应用

大量的这些观点会应用到一个真正的 DSP 例子里，假若这样就能被应用到 FFT 设计实现中，则这能用许多的方法实现。在 Meyer - Baese（2001）里可以看到许多 FPGA 的实现信息。想到 N 个复杂数据点的 DFT，$x(n)$被定义为

$$X_k = \sum_{n=0}^{N-1} x(n) W_N^{nk} \quad k = 0, 1, \cdots, N-1 \tag{13-9}$$

式中，W_N 是旋转因子，$W_N = e^{-j(2\pi/N)}$。FFT 可以说是来自于它，正如在 Bi 和 Jones（1989）中提出的，其中涉及 DFT 表达式的改写。到目前为止最常见及广泛使用的是 Cooley - Tukey 算法（Cooley 和 Tukey1965），它通过索引映射的使用将算法复杂性从 $O(N^2)$ 降低到了 $O(N\log N)$，其中用到了转换参数中的对称性和周期性。直到仅仅需要 4 点 DFT，基 4 的实现才递归地分解了这个算法。这个结果被结合到 N 点变换的以计算中。FFT 的计算使用蝶形（图 13-13a）和完美的随机网络（图 13-13b）。算法的全局递归本质表明自身是全局的互连，这需要不规则的路由，其中数据常规地被传到并不邻近的处理部件中（PEs，Stone 1989）。基 4 算法的表达式如下：

$$X_k = \sum_{n=0}^{\frac{N}{4}-1} x(n) W_N^{nk} + W_N^{\frac{Nk}{4}} \sum_{n=0}^{\frac{N}{4}-1} x\left(n + \frac{N}{4}\right) W_N^{nk} + W_N^{\frac{Nk}{2}} \sum_{n=0}^{\frac{N}{4}-1} x(n + N/2) W_N^{nk} + W_N^{3Nk/4} \sum_{n=0}^{\frac{N}{4}-1} x\left(n + \frac{3N}{4}\right) W_N^{nk} \tag{13-10}$$

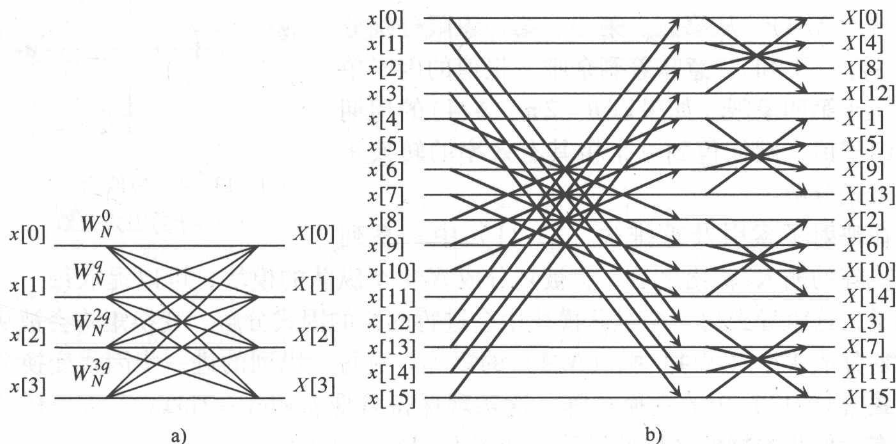


图 13-13 FFT 结构

a）基 4 的蝶形 b）16 点基 4 的 FFT

典型的，更大数量点的 FFT 创建自小的 FFT 模块，但是固有的不规则性会影响速度并且对功耗也有影响。计算的选择性分解是可能的，其中涉及如式（13-11）所示的 16 点 DFT 矩阵结构中重复图样的识别。

$$
\begin{bmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \\ X_8 \\ X_9 \\ X_{10} \\ X_{11} \\ X_{12} \\ X_{13} \\ X_{14} \\ X_{15} \end{bmatrix}
=
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & W & W^2 & W^3 & -j & W^5 & W^6 & W^7 & -1 & -W & -W^2 & -W^3 & -j & -W^5 & -W^6 & -W^7 \\
1 & W^2 & -j & W^6 & -1 & W^2 & j & -W^6 & 1 & W^2 & -j & W^6 & -1 & -W^2 & j & -W^6 \\
1 & W^3 & W^6 & -W & j & -W^7 & W^2 & W^5 & -1 & -W^3 & -W^6 & W & -j & W^7 & -W^2 & -W^5 \\
1 & -j & -1 & j & 1 & -j & -1 & j & 1 & -j & -1 & j & 1 & -j & -1 & j \\
1 & W^5 & -W^2 & -W^7 & -j & -W & -W^6 & W^3 & -1 & -W^5 & W^2 & W^7 & j & W & W^6 & -W^3 \\
1 & W^6 & j & W^2 & -1 & -W^6 & -j & -W^2 & 1 & W^6 & j & W^2 & -1 & -W^6 & -j & -W^2 \\
1 & W^7 & -W^6 & W^5 & j & W^3 & -W^2 & W & -1 & -W^7 & W^6 & -W^5 & -j & -W^3 & W^2 & -W \\
1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1 \\
1 & -W & W^2 & -W^3 & -j & -W^5 & W^6 & -W^7 & -1 & W & -W^2 & W^3 & j & W^5 & -W^6 & W^7 \\
1 & -W^2 & -j & -W^6 & -1 & W^2 & j & W^6 & 1 & -W^2 & -j & -W^6 & -1 & W^2 & j & W^6 \\
1 & -W^3 & W^6 & W & j & W^7 & W^2 & -W^5 & -1 & W^3 & -W^6 & -W & j & -W^7 & -W^2 & W^5 \\
1 & j & -1 & -j & 1 & j & -1 & -j & 1 & j & -1 & -j & 1 & j & -1 & -j \\
1 & -W^5 & -W^2 & W^7 & -j & W & -W^6 & -W^3 & -1 & W^5 & W^2 & -W^7 & j & -W & W^6 & W^3 \\
1 & -W^6 & j & -W^2 & -1 & W^6 & -j & W^2 & 1 & -W^6 & j & -W^2 & -1 & W^6 & -j & W^2 \\
1 & -W^7 & -W^6 & -W^5 & j & -W^3 & -W^2 & -W & -1 & W^7 & W^6 & W^5 & -j & W^3 & W^2 & W
\end{bmatrix}
\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \\ x_{10} \\ x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{15} \end{bmatrix}
\tag{13-11}
$$

图 13-14 所示为如何反映虚轴的左右根值和实轴的上下根值。根值为 1，$-j$，-1 和 j 的角度 q 为 $2\pi/(N/4)$。从第三、第二、第一象限分别分解出 $-j$，-1 和 j，意味着剩余唯一根值的因子单独地位于第四象限。如果以 $\theta=2\pi/(N/4)$ 的周期来分组根值，则能得到一个极其有效率的转换矩阵的因式分解。

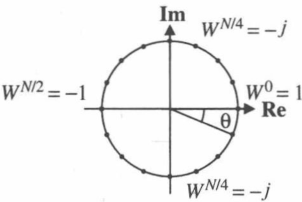


图 13-14 根值为 -1，j 和 $-j$ 的因式分解

在映射了索引及重排式（13-11）中一系列 DFT 矩阵的输入/输出之后，会被划分成 $N/4$ 个纵队的模块，可以提取每一行的公因子，这就导致每一个纵队模块中会进行同样的因式分解。变换矩阵会被从行向量划分成 4 个（$N/4$）乘（$N/4$）的矩阵。对每一纵列的进一步因式变换会产生如式（13-12）中的变换矩阵。这实现保持着像 Cooley – Tukey 算法一样的计算效率，但是数据的区域性增强，如图 13-15 所示。

$$
\begin{bmatrix} X(k_0,0) \\ X(k_0,1) \\ X(k_0,2) \\ X(k_0,3) \end{bmatrix}
=
\begin{bmatrix}
1 & 1 & 1 & 1 \\
1 & -j & -1 & j \\
1 & -1 & 1 & -1 \\
1 & j & -1 & -j
\end{bmatrix}
\begin{bmatrix} a_{k1}(0) \\ W^{k1} a_{k1}(1) \\ W^{2k1} a_{k1}(2) \\ W^{3k1} a_{k1}(3) \end{bmatrix}
\tag{13-12}
$$

这种架构最初作为 20 世纪 90 年代实施的一个主要芯片设计项目的一部分来

开发，其涉及女皇大学的 Belfast，Snell 和 Wilcox，以及以开发 64 点 FFT（QFFT 64）处理器为目的的 BBC。工作的重点是为了研究这种规律性是否会转变到 FPGA 的实现中，因此一个 64 点 FFT 架构是用 VHDL 编码的，而且与一个出自 Amphion 和 XFFT64 的 64 点软 IP FFT 核的 AFFT64 相比，一个 64 点的 FFT 核一般是来自 Xilinx's Coregen IP 库的。后两种设计被高度参数化而且能够在不同的模式下运行，但是两者在图 13-13a 所示中都仿佛是基于一个相同的架构。表 13-4 已经从 Virtex－2Pro 平台的结果中得到了开发，这也是为了能够通过使用实验连接线及真正的功率测试方法来测量功耗。随着核心在不同的速度下运行，大量不同的核心也被用于每个配置，也就是在 800 MSPS 下的一个 64 点 QFFT、4 个 64 点 XFFT 和 4 个 64 点 AFFT 的核心，以及在 2.4 GSPS 下的 3 个 64 点 QFFT、12 个 64 点 XFFT 和 12 个 64 点 AFFT 的核心。

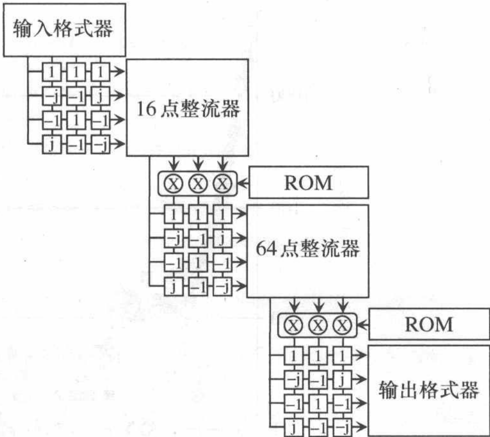


图 13-15　64 点 FFT 架构

表 13-4　能量功耗和对于 800MSPS 的资源的使用

	功率		资源		
	数据 1	数据 2	部分	LUT	Mult18 × 18
QFFT64	1029	945	1950	2966	18
XFFT64	1616（36%）	1496（37%）	4991（61%）	6571（55%）	242（5%）
AFFT64	4044（75%）	3288（71%）	9697（80%）	15871（81%）	0
QFFT64	1029	945	1950	2966	18
XFFT64	1616（36%）	1496（37%）	4991（61%）	6571（55%）	242（5%）

图 13-16 所示为不同网络长度的转换率。如图 13-12 所示的乘法器例子，架构的规律性已经导致了网络长度变得更短，正如通过在 x 轴上的电容量所表述的。此外，这些网络的转换急剧得变少，这代表了对核心而言主要的节约功耗的做法。

除了常规结构外，都是设计成流水线式的，它在所有的设计上都能缩减网络长度。为了进一步最小化互连，参数会得到预先的计算并且存储于 PE 中的分布式 RAM。输入、输出和整流器元件由一系列时延和整流数据选择器组成。时延线作为移位寄存器 LUT（SRL）得到实现，它使得变化的时延

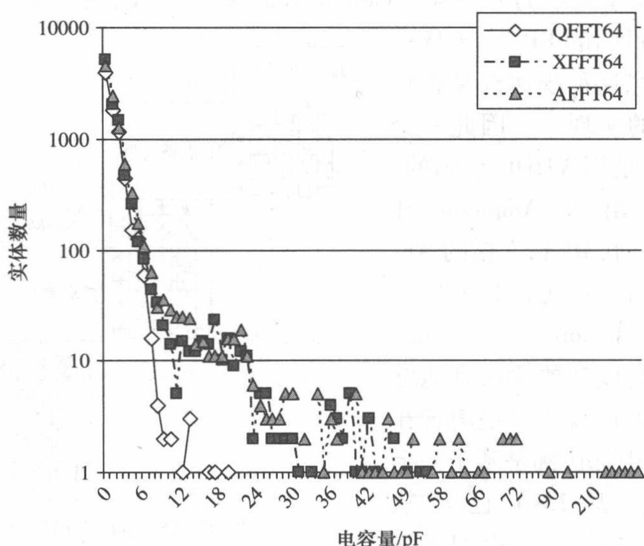


图 13-16 64 点 QFFT、64 点 XFFT 和 64 点 AFFT 在 800 MSPS 生产量下的信号容量

长度没有互连处罚。

13.11 总结

本章介绍了 FPGA 中功耗的来源，包括静态和动态的功耗。同时提出了一些可以降低静态功耗的技术，大多数技术都集中在动态功率的降低上，因为这代表了对设计者有利的最主要技术，这都归因于 FPGA 预制的本质。大量的技术包括数据重排、固定系数的功能性，流水线式的使用及规律的强制性是很突出的。这些都被应用到 64 点 FFT 处理器的设计上了。

还有很多其他技术值得考虑，比如存储器架构的发展。FPGA 的一个重要的优点是局部存储的有效性。作为特色，这还没被开发利用，是因为伴随着多重存储器资源运行会很困难。但是，开发高级设计方法论来使得在最后的结果中构造局部存储器成为可能，这种做法有着很大的吸引力，因为比起这里提到过的一种最优化，它代表了一种更加强有力的区域性最优化。此外，这能从数据流层中分离出来，正如 Fischaber 等（2007）中表现出来的。作者的观点是在下一个十年里，这将会逐渐成为一个研究领域。

参考文献

Raghunathan A, Dey S and Jha NK (1999) Register transfer level power optimization with emphasis on glitch analysis and reduction. *IEEE Trans. CAD* 18(8), 114–131.

Bi G and Jones EV (1989) A pipelined fft processor for word-sequential data. *IEEE Transactions on Acoustic, Speech, and Signal Processing* 37(12), 1982–1985.

Bowman K, Austin L, Eble J, Tang X and Meindl J (1999) A physical alpha-power-law mosfet model *IEEE Journal of Solid-State Circuits*, 32, pp. 1410–1414.

Chandrakasan A and Brodersen R (1996) *Low Power Digital Design*. Kluwer.

Chen CS, Hwang TT and Liu C (1997) Low power fpga design – a re-engineering approach *Proc. 34th Design Automation Conference*, pp. 656–661.

Chow CT, Tsui LSM, Leong PHW, Luk W and Wilton S (2005) Dynamic voltage scaling for commercial fpgas *Int. Conf. on Field Programmable Technology*, pp. 215–222.

Cooley J and Tukey J (1965) An algorithm for the machine calculation of complex Fourier series. *Mathematics of Computation* 19, 297–301.

Erdogan A and Arslan T (2002) Implementation of fir filtering structures on single multiplier dsps. *IEEE Trans. Circuits and Systems II* 49(3), 223–229.

Fischaber S, Woods R and McAllister J (2007) Soc memory hierarchy derivation from dataflow graphs *IEEE Workshop on Signal Processing Systems*, pp. 469–474.

Gonzalez R, Gordon B and Horowitz M (1997) Supply and threshold scaling for low power cmos. *IEEE Journal of Solid-State Circuits*, 32(8), 1210–1216.

Hui C, Ding TJ, Woods RF and McCanny JV (1996) A 64 point Fourier transform chip for video motion compensation using phase correlation. *IEEE Journal of Solid State Electronics* 31(11), 1751–1761.

IRTS (2003) *International Technology Roadmap for Semiconductors*, 2003 edn. Semiconductor Industry Association.

Keane G, Spanier JR and Woods R (1999) Low-power design of signal processing systems using characterization of silicon ip cores. *33rd Asilomar Conference on Signals, Systems and Computers*, pp. 767–771.

Kim N, Austin T, Baauw D, Mudge T, Flautner K, Hu J, Irwin M, Kandemir M and Narayanan V (2003) Leakage current: Moore's law meets static power. *IEEE Computer* 36(12), 68–75.

Kung HT (1982) Why systolic architectures?. *IEEE Computer* 15(1), 37–46.

Kung HT and Leiserson CE (1979) Systolic arrays (for vlsi). *Sparse Matrix Proc. 1978*, pp. 256–282.

Lamoureux J and Wilton S (2003) On the interaction between power-aware fpga cad algorithms *Proc. Int. Conf. on Computer-aided Design*, pp. 701–708.

McKeown S, Fischaber S, Woods R, McAllister J and Malins E (2006) Low power optimisation of dsp core networks on fpga for high end signal processing systems *Proc. Int. Conf. on Military and Aerospace Programmable Logic Devices*.

Mead C and Conway L 1979 *Introduction to VLSI Systems*. Addison-Wesley Longman, Boston.

Meyer-Baese U (2001) *Digital Signal Processing with Field Programmable Gate Arrays*. Springer, Germany.

Roy K, Mukhopadhyay S and Meimand H (2003) Leakage current mechanisms and leakage reduction technqiues in deep-submicron cmos circuits. *Proc. IEEE* 91(2), 305–327.

Stone HS (1989) Parallel processing with the perfect shuffle. *IEEE Trans. on Circuits and Systems* 36, 610–617.

Wilton SJE, Luk W and Ang SS (2004) The impact of pipelining on energy per operation in field-programmable gate arrays *Proc. Int. Conf. on Field Programmable Logic and Application*, pp. 719–728.

Wolf W (2004) *FPGA-Based System Design*. Prentice-Hall, New Jersey.

Xilinx Inc. (2007) White paper: Virtex-5 fpgas 'power consumption in 65 nm fpgas'. Web publication downloadable from www.xilinx.com.

第14章　最后陈述

14.1　引言

本书已经探讨了许多应用FPGA创建DSP系统解决方案所需的技术。它们有些被封装在设计工具中，然后应用到大量设计实例中。除了要在面积、速度和吞吐量速率方面取得有效的FPGA实现外，这本书还涵盖了低功耗的关键技术，主要涉及大量以减少动态损耗为目的的技术说明。这些方法的关键是利用FPGA底层资源成功推导出有效的电路结构，以匹配最佳的计算和通信应用的需求量。通常情况下，这涉及使用更高级的并行和流水线操作。

如果基于FPGA的DSP系统设计是自动实现的，则需要结合第9章描述的基于HDL或C语言设计工具（如开头提到的Synplify DSP工具），或如第11章中所概述的更高级的设计环境。如果这能成功，则任何高级方式都应该允许设计师从高级描述创造高效FPGA应用，或者纳入现有的IP核系统中，但这很可能还需要多年的设计时间去达到这种高级设计。

本章的目的是关注一些不得不被忽略，或没有足够详细介绍的问题。14.2节的重构FPGA系统导致了后一类问题，因为FPGA的可编程特质提供了一个有趣的平台，允许此类系统的实现。这需要额外的注释，以及14.2节和14.3节分别提到的FPGA上的浮点型硬件的实现和内存架构的创建。14.4节还试图确定未来的发展趋势，概述FPGA开发商未来的挑战。

14.2　可重构系统

传统上，可编程性属于微处理器的领域，它执行一系列指令完成系统所需行为，通过改变指令能更改系统功能。它的优点是不需要任何电路修改，但这是有限的，因为它不能为客户自定义硬件提供同样优良的处理性能和电源消耗。问题出在相当一部分的电路是用于存储和控制现代微处理器电路的，这种开销需要让计算任务大量重用微处理器的小型活动部分，即功能单元（Function Unit，FU）。

从概念上讲，FU都需要评估描述系统的操作。然而，在实践中，处理器需要在FU之间，以及内存和FU之间移动数据。当高速、多端口存储器可以放置在FU附近时，情况会变得更为复杂。为了解决这个问题，处理器内部放置有内

存分层。速度最快、消耗最大的内存（寄存器排列）形成内存层次结构的顶层，置于系统核心的最近处来存储数据。每向下一层，内存能力增加，访问速度变慢。不同层之间的数据迁移和管理则是处理器所需的另一项重要任务。

发展这种可编程体系结构的结果是，在一次运算中大量的晶体管不会执行有用计算。在硅工艺日益成熟的情况下，减少对大内存块的瓶颈压力还有很大的需求。关于系统内存，有种效应称为内存墙，被 Flynn 等人（1989）建议作为指示内存访问时间与处理器的周期时间的比值，它将随着技术改进而增加。

这倾向于表明结构解决方案如何推导需要大的变化。在某种程度上，20 世纪 80 年代在那些对互连线时延影响的致命预测中已经看到了这些，例如基于脉动并行计算阵列的结构（Kung 和 Leiserson 1979，Kung 1988）；一项重大发展是由英特尔和卡内基·梅隆大学 H T Kung 联合开发的命运多舛的多重处理超级计算机 iWarp。目标是根据经典脉动阵列理论，把本地化的内存和最近通信连接在单个微处理器中建成一个并行计算节点。虽然这一概念很适合于高度计算，如矩阵和矢量计算，但对于不规则的数据计算，它工作得不够好。

然而，可重置计算不受这种限制的影响，它的主要魅力是基于计算需要改变硬件实现。它针对复杂系统，目的是使复杂计算分解成可编程硬件。在本书的前几节中可以看到，可重构置计算使控制导向方面更多地映射到一个更合适的平台，即处理器的加速可见。这推断出连接单个或 FPGA 阵列的可编程处理器的使用。

14.2.1 FPGA 可编程技术的相关性

在早期的 FPGA 中，出现了大量不同的编程技术，特别是 E^2PROM 技术、熔丝技术及 SRAM 技术。SRAM 编程技术带来了崭新的操作模式，即功能性可随操作改变，或换句话说——可重构。这既可以在正常模式的下载时间内静态完成，也可以作为常规操作模式的一部分动态完成。基本上，FPGA 包括大量的可编程逻辑、寄存器、内存块和可配置以不同方式来实现各种功能的专用处理模块。FPGA 可以被认为是像一个智能内存设备，硬件结构的“状态”从处理器下载到 FPGA 器件上。此 FPGA 配置被用来对传入的数据作出对应操作。通过重写 FP-GA 器件的不同数据改变操作功能。因此，与其把数据存储到内存设备写读，不如将数据存储到 FPGA 器件上，FPGA 对数据的处理功能是可以通过编程改变的，由此从 FPGA 器件得到处理结果。

这被指定包括两个不同阶层，即执行如上文所述操作并存储可用内存数据所需信息的逻辑层，以及包含所需编程信息的 SRAM 配置层。因此，与逻辑层交互是运作的正常模式，而编程 SRAM 配置层定义了配置模式。此编程模式若想具有吸引力，则意味着它要使硬件编程最好地满足计算需求，在需要时理想地达到需求。

这一概念仍然具有关键的吸引力，因为它意味着可用硬件资源可匹配大多数功能要求的有效实现，但这也带来了一些挑战。包括在如 DSP 环境下配置硬件时间的影响，数据不断输入，因而必须存储起来。另外，还有在处理时间上的影响，如图 14-1 所示（Herson 等 2001），重置时间 T_R 可对性能产生影响，尽管 FPGA 的性能率更优。因此是得失相当平衡的时间 T_B，它成为执行重置的优势。随着设备状态不断改变，系统的基本可靠性不再是问题；而问题在于确认硬件是否配置正确。

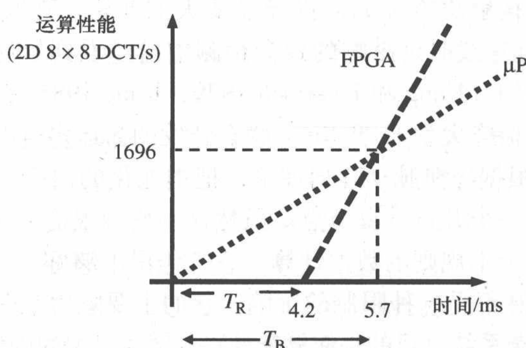


图 14-1 重新配置时间对 8 ×8 的 2D DCT 的影响

14.2.2 现有重置计算

如今已有大量的重置计算平台，并有很好的评价（Bondalapati 和 Prasanna 2002，Compton 和 Hauck 2002，Todman 等 2005）。重置系统的分类由 Compton 和 Hauck（2002）最初强调，并由 Todman 等人（2005）修改。图 14-2 所示为 Todman 等人的简化版图解。此图显示了 FPGA 形式的重置单元如何为常规的 CPU 提供 CPU/FPGA 配置。

图 14-2a 所示的第一类可重置单元被视为外部处理单元。大量这种例子都有主要 FPGA 供应商配置，也有 Celoxica 等公司的配置。第二和第三类使用可重置单元作为一个专用协处理器单元，或只连接到总线，或同时连接 CPU 和总线。研究例子包括 RAPid（Ebeling 等 1996）和 Piperench（Laufer 等 1999）。第四类包含可重置单元或 CPU 结构，或以结构的形式更改单个处理器的相互联系，或可能包括改变子单元功能的重置，或二者皆有。一个重置例子是 Elixent（2005）的可重置算法处理（Reconfigurable Algorithm Processing，RAP）技术。

随着第 5 章中所述 FPGA 技术的演变，这种比较现在有点过时。现在的 FPGA 既可用软件也可以用底层硬件微处理器实现。这意味着，在某些商业和高性能的计算解决方案中，把 FPGA 作为硬件加速器仍是可以的，但也出现了结合

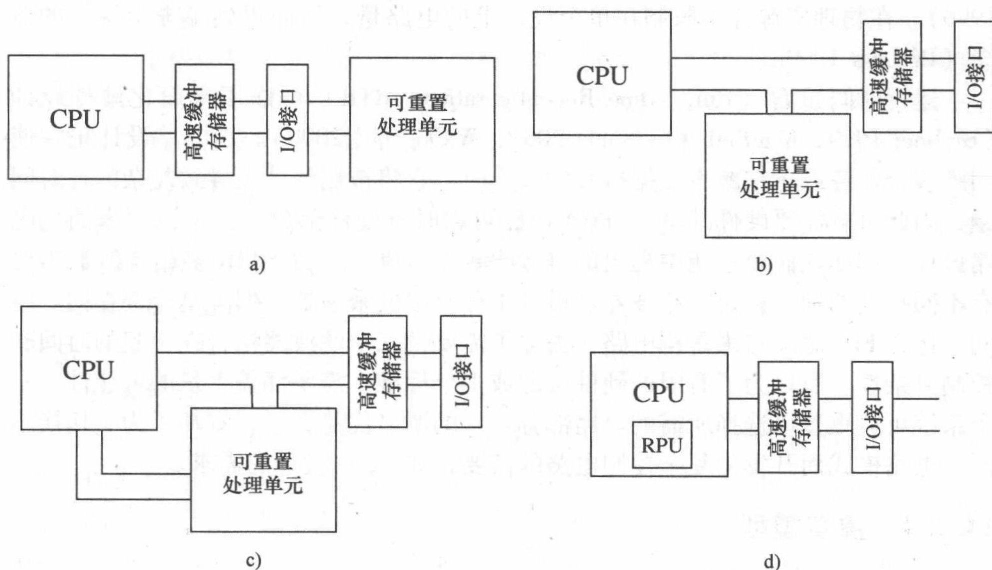


图 14-2 可重置系统的分类

a）外部独立处理单元 b）附加处理单元 c）协同处理器 d）可重置功能单元

FPGA 结构内核心的概念。然而，考虑到通信/计算硬件的比率，在速度和成本方面比性能带来影响的方法，许多模型处理的问题仍然适用。

14.2.3 重置的实现

有很多的方法可以实现重新编程，每种方法都有微妙但重要的差异。首先就在于如何在设计中使用重新编程，允许该电路纳入一定程度的灵活性。如果在任何特定时刻纳入许多部分冗余组件，则一个大型电路将很快变得非常大。Mac Bath和 Lysaght（2001）指出可重复编程器件中有一个可重复编程的核是不必要的开销，并提出把这些电路分类作为可编程多功能核（Programmable Multi-function Cores, PMC）。这也可以帮助 FPGA 缩小与 ASIC 的表现差距；在 ASCI 中，电路以一种提供多功能性的方式获得提升，允许将所有可能的情况概括为一种实现，从而交换到灵活性的区域和速度。由于 FPGA 结构用本身的性质支持重置，因此这些资源可以用于更改电路（的状态），以在任何时间运行最佳电路。电路越有针对性，越可能修正以保持优势，但在电路区域和数据吞吐量方面还有更多好处。把 FPGA 的结构作为电路状态变化过程的一部分使用，可以减少 21% 的区域，并增加 14% 的速度。

运行时间重置

FPGA 执行逻辑操作的能力有限，但可以用重置来绕过它。通过分时的 FP-GA 逻辑，FPGA 可以实现超越硬件区域能力的电路操作（Lysaght 和 Stockwood

1996）。在物理资源并不限制在单个设备上的电路量，从而得到了虚拟硬件的概念（Brebner 1999）。

运行时间重置（Run - time Reconfiguration，RTR）可以用来简化硬件设计（Brebner 1999，MacBeth 和 Lysaght 2001，Walke 等　2000）。当电路设计正在使用跨多个任务或正在减少数据折叠的大小时，它将被制约从而导致复杂的计时问题，因此可能需要硬件共享。FPGA 电路可以时分硬件部分，这可以用来简化电路设计。例如在假设系统中提出的（Walke 等 2000），三个 FIR 多相滤波器需要有不同长度的抽头；每个滤波器被设计十分有限的乘法器，但不是全部在同一时间；选择 FIR 滤波器来重构电路，去除了需要把三个滤波器结合在一起的时间和控制复杂性。所以为了有限的硬件资源或多个标准，需要涵盖大量的可能性，一个系统可以重置以选择所需的电路部分。它的明显优势在于，省却了为了切换不同的电路模式而开发的复杂控制电路的需要，如 ASIC 设计的需求。

14.2.4　重置模型

出现了大量的重置模型。

1. 单一环境

传统模式的商用 FPGA 是单一环境，只允许通过加载全新的配置来改变电路。这种类型的模型不适合 RTR，因为在固件中进行更改需要关联时间。写入数据的速度被从源到 FPGA 的配置带宽限制。这可以通过设备上接口的宽度（引脚）和速度，以及接口与系统相连的方式确定。虽然重置数据被写入设备中，但当前电路仍不能用。其他功能已提出，并要克服这种带宽限制。

2. 部分重置

商用 FPGA 的重置包括 Altera 的 Stratix® III 系列和 Xilinx 的 VirtexTM-5 FPGA 系列。由于不分科独立更改，因此比起单一环境 FPGA，这些设备的 RTR 范围更大。由于重置部分变小，制造变化的花费减少，因此不需要重大开销就能频繁改动。这种器件的一个特点就是不断的操作。当 FPGA 的部分重置时，隐匿了重置需要的时间。

3. 多元化环境

对于多元化环境设计的研究有很多（DeHon 1996，Scalera 和 Vazquez 1998，Trimberger 等　1997），但迄今为止，并未有可行的商用产品。这些设备通过存储多于一面以上的片上重置数据，来解决转移和重构数据的问题。重置的过程就是选择其中一面来驱动逻辑重置。环境之间转换只需少数时钟周期。多元环境允许后台加载配置数据，因为配置数据可以加载到处于非活动状态的环境中。

这种装置的一个可能问题在于一个典型的系统应用所需的环境数量，这可能是大于可用硬件和环境间的数据共享。第 2.6.2 节的 SCORE 结构与乘数环境 FPGA 有些相似，但允许更改环境使用的资源量。

4. 流水线重构

流水线重构模型表示了作为流水线模块的设备重构，并可被看作是一个可调重构 FPGA（Luk 等 1997），它被视为流水线数据通路，每个阶段整体重构。这允许重构和执行的累加。所需的重构被分为按顺序加载的重构阶段。在每个阶段编好程后，立即开始运作，因此一个阶段的配置正是数据流的前一步。一旦设备空间不足，它就开始交换内存在 FPGA 上储存时间最长的程序，在下个阶段取代它们。这允许超过设备物理资源的应用降低吞吐量继续运行。Piperench（Laufer 等 1999）是这种类型模型的一个例子。在这种情况下，作者指出向前兼容性的优势，附加设备可以只是增加阶段数，来保留相同的阶段。

5. 配置缓存

图 14-3 所示为 FPGA 数组和一个重置数据存储设备的 FPGA 抽象模型。由于不能提供所需的带宽，因此配置数据的运动会导致电路处理的长时间中断。在这种情况下，可以使用图 14-3 所示的预取和缓存，减少“突发性”的数据移动（Hauck 等 1999）。接近 FPGA 数组的缓存数据，如片上，可以通过窄带预取缓存，来使用一个高带宽通道。然而，当获取不正确的数据时，预取会出现条件分支的问题。

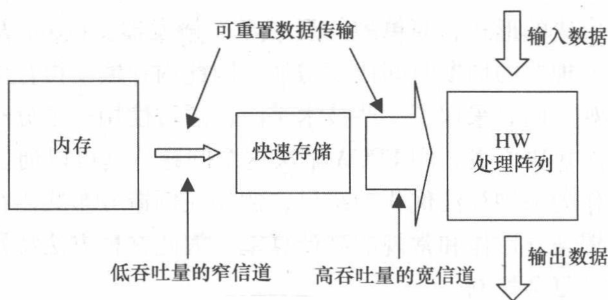


图 14-3 用于绕过可重置数据传输的带宽限制的预取和缓存

6. 配置压缩

配置压缩减少了要移动的数据量，因此通过使用数据压缩削减重置时间。图 14-4 所示为该模型的抽象视图。预取时，解压硬件与 FPGA 数组中的连接远大于它与数据存储的连接。这个概念最初是为 Xilinx XC6200 FPGA 技术开发的，它具有通配符功能，同时允许地址和数据的值写入多个位置。开发这个设备专门用来支持动态重置，并认识到有必要在同一时间改变设备的部件，这就是通配符功能。Hauck 等人（1999）指出可以以一种方法压缩数据流，使 XC6200 的通配符硬件解压数据流。Hauck 等人（1999），Li 和 Hauck（2001）继续考虑可以用于其他 FPGA 的低开销压缩方法。

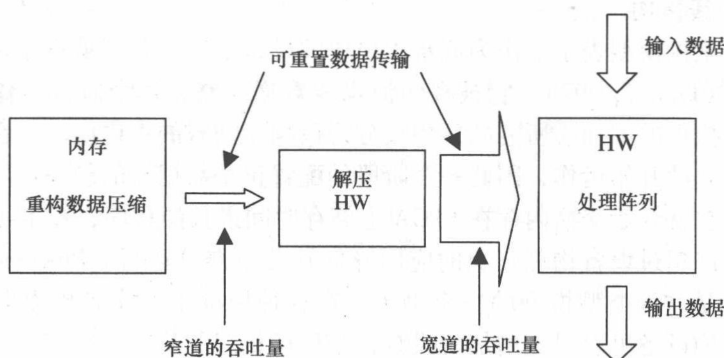


图 14-4 用于绕过重新配置数据传输的带宽限制的压缩

14.3 内存体系结构

当考虑实现 DSP 系统时，FPGA 的主要吸引力在于对并行和流水线操作的支持。然而，受到重视的一个因素是一系列不同尺寸的并行存储器的可行性，无论这是分布式内存模块的形式、简单的 LUT 或单个触发器。Flynn 等人（1989）强调，由于内存到处理器的周期时间比率增加，因此内存墙使得存取固态计算机架构的未来不容乐观。同时采取了一些技术手段，通过使用一个分布式内存结构开发一种高度并行的解决方案，用 FPGA 解决这个问题。这可以通过特定分布式存储结构推导，或作为一种算法优化的结果，例如在创造分布式内存影响下的流水线应用。由于数据独立运作和常规的高计算率，因此这种方法特别适合 DSP，从而使并行结构不受复杂控制。

这就表明，FPGA 提供了一个有用的平台，用于开发以内存，而不是以计算为导向的新型计算机架构。文本已清楚表明不同的电路架构是如何发展为不同 DSP 算法的。结构性派生通常是被创造必要的计算资源以满足系统的详细规格参数所驱动。有充分的理由表明以内存为导向的架构的开发需要更详细的研究，其基本规则被开发用于创造一些 DSP 算法的内存需求（Kapasi 等 2002）。这在内存提升到满足算法级需求的想象处理器中有某种程度上的预见。第 6，8，9 章的细节 FPGA 示例中，或是因为缺乏触发器资源，或是与相关资源的选择有关，不同的内存，即 SRL 形式的 LUT，被优先选择用于触发器来提供更多有效应用延迟链。然而，这往往是一个好的设计决定，而不是有意识提升内存为导向的体系结构的需求。Fischaber 等人（2007）的建议可从数据流水平进行内存的设计工作。

14.4 对浮点计算的支持

在早期 FPGA 中一个有意识的决定是首先引入了可扩展加法器结构，之后的版本是引入专用乘法器复杂性，例如 Altera 的 Stratix®III 系列和 Xilinx 的 Virtex™-5 FPGA 系列，这些已经真正地影响到了 DSP 系统的 FPGA 的使用。因为所需的计算率极高，所以分布式内存的可行性推动使了用 FPGA 进行超级计算的溢出。Cray，SGI 和 Nallatech 等供应商一系列广泛的解决方案在大量硬件平台上的可行性，会在高级计算上有很好的表现。然而 Craven 和 Athanas（2007）的工作表明，即使用本书中强调的很多方法，高级计算应用可达到的性能仍被限制。

一个关键的原因是 FPGA 中缺乏适当的浮点支持，即使是 Craven 和 Athanas（2007）也避免在工作中使用浮点算术而使用固点硬件。表 14-1 重述了第 3 章的数字，突出了在 FPGA 中使用浮点算术的问题。如果字长效应小于 18 位，则执行一个固点将只需一个 DSP48，不需要很多表中所述的大量触发器。执行数据选择、舍入和正常化必定需要额外的硬件来执行，如图 3-10 所示。如果 FPGA 被广泛应用于超级计算机，则对功能的支持需要在今后的 FPGA 版本中对流水线有更好的高级编程支持。

表 14-1 使用 Xilinx Virtex-4 FPGA 技术实现不同浮点操作的分区和速度数据

函数	DSP48	LUT	FF	速率（MHz）
乘法器	4	799	347	141.4
加法器		620	343	208.2
倒数	4	745	266	116.5

14.5 FPGA 未来的挑战

当今的商业 FPGA 产品令人印象非常深刻。最近的设备代表了推动使用硅技术，并更多地用于内存，作为复杂设备来测试未来硅技术的高度复杂平台。然而，会有一些特别影响到 FPGA 的挑战。

技术扩展提供了许多优点，但它也突出了一个特别棘手的问题，那就是互连晶体管的延迟比率。表 14-2 中显示一些例子，以及各种技术晶体管的延迟比率（Davis 等 2001）。在这个表中，最小扩展表示如果不考虑任何措施产生的结果，则反向扩展指育肥电线的影响，从而扭转扩展过程。反向比例将被用来抵消互连增加的阻力影响，见表 14-2。此时间问题会导致布线密度急剧下降，从而降低集成水平，从区域的角度导致总线驱动结构更加低效。由于 FPGA 高度依赖

可编程路由，因此这两种方法都有问题。一个是最小比例，将大大减慢系统；而反比例，将有一个主要区域影响。看来这种效应的影响将增加 FPGA 上更大的异质块的使用。

表 14-2 互连延迟对技术测量的影响

技术	MOSFET 转换延迟	1mm 互连的内部延迟	
		最小测量值	最大测量值
1.0μm（Al，SiO_2）	20	5	5
1.0μm（Al，SiO_2）	5	30	5
35μm（Cu，low k）	2.5	250	5

逐步扩展带动了半导体行业，允许供应商提供更快、更便宜，且功能不断增加的电路。互连延迟影响的增加，带来一个更严重的问题，即过程变化的影响。到 2018 年 40nm 的技术将转为子 10nm（半导体行业协会 2005），现在设备特性变异是提供下一代 SoC 系统，包括 FPGA 的一个重大挑战。其他问题可能采取最坏的案例，然后使用其他办法来克服局限性，但过程变异问题产生的原因，需在下一代扩展中进行调查。

参考文献

Bondalapati K and Prasanna V (2002) Reconfigurable computing systems. *Proc. IEEE* 90(7), 1201–1217.

Brebner G (1999) Tooling up for reconfigurable system design *Proc. IEE Coll. on Reconfigurable Systems*, pp. 2/1–2/4.

Compton K and Hauck S (2002) Reconfigurable computing: a survey of systems and software. *ACM Computing Surveys* 34(2), 171–210.

Craven S and Athanas P (2007) Examining the viability of fpga supercomputing. *EURASIP Journal on Embedded Systems* 1, 13–13.

Davis J, Venkatesan R, Kaloyeros A, Beylansky M, Souri S, Banerjee K, Saraswat K, Rahman A, Reif R and Meindl J (2001) Interconnect limits on gigascale integration (gsi) in the 21st century. *Proc. IEEE* 89, 305–324.

DeHon A (1996) Dynamically programmable gate arrays: A step toward increased computational density *Proc. 4th Canadian Workshop on Field-Programmable Devices*, pp. 47–54.

Ebeling C, Cronquist DC and Franklin P (1996) Rapid: Reconfigurable pipeline datapath *Proc. 6th Int. Workshop on Field-Programmable Logic and Compilers*, pp. 126–135.

Elixent (2005) Reconfigurable algorithm processing (rap) technology. Web publication downloadable from http://www.elixent.com/.

Fischaber S, Woods R and McAllister J (2007) Soc memory hierarchy derivation from dataflow graphs *IEEE Workshop on Signal Processing Systems*, pp. 469–474.

Flynn MJ, Hung P and Rudd K (1989) Deep-submicron microprocessor design issues. *IEEE Micro* 19, 11–22.

Hauck S, Li Z and Schwabe EJ (1999) Configuration compression for the xilinx xc6200 fpga. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems* 18(8), 1107–1113.

Heron J, Woods R, Sezer S and Turner RH (2001) Development of a run-time reconfiguration system with low reconfiguration overhead. *Journal of VLSI Signal Processing, Special issue on Re-configurable Computing* 28(1/2), 97–113.

Kapasi U, Dally W, Rixner S, J.D. O and Khailany B (2002) Virtex5.pdf *Proc. 2002 IEEE Int. Conf. on Computer Design: VLSI in Computers and Processors*, pp. 282–288.

Kung HT and Leiserson CE (1979) Systolic arrays (for vlsi) *Sparse Matrix Proc. 1978*, pp. 256–282.

Kung SY (1988) *VLSI Array Processors*. Prentice Hall, Englewood Cliffs, NJ.

Laufer R, Taylor R and Schmit H (1999) PCI-Piperench and the SWORDAPI – a system for stream-based reconfigurable computing. *Proc. IEEE Symp. on Field-programmable Custom Computing Machines*, Napa, USA, pp. 200–208.

Li Z and Hauck S (2001) Configuration compression for virtex fpgas *Proc. IEEE Conf. on FPGA-based Custom Computing Machines*, pp. 147–159.

Luk W, Shirazi N, Guo SR and Cheung PYK (1997) Pipeline morphing and virtual pipelines. *7th Int. Workshop on Field-programmable Logic and Applications*, pp. 111–120.

Lysaght P and Stockwood J (1996) A simulation tool for dynamically reconfigurable field programmable gate arrays. *IEEE Trans. on VLSI Systems* 42(2), 381–390.

MacBeth J and Lysaght P (2001) Dynamically reconfigurable cores *Proc. IEEE Conf. on FPGA-based Custom Computing Machines*, pp. 462–472.

Maxfield C (2004) *The Design Warrior's Guide to FPGAs*. Newnes, Burlington.

Scalera SM and Vazquez JR (1998) The design and implementation of a context switching fpga *Proc. IEEE Conf. on FPGA-based Custom Computing Machines*, pp. 78–85.

Semiconductor Industry Association (2005) International technology roadmap for semiconductors: Design. Web publication downloadable from http://www.itrs.net/Links/2005ITRS/Design2005.pdf.

Todman T, Constantinides G, Wilton S, Cheung P, Luk W and Mencer O (2005) Reconfigurable computing: architectures and design methods. 152(2), 193–205.

Trimberger S, Carberry D, A. J and Wong J (1997) A time-multiplexed fpga *Proc. IEEE Conf. on FPGA-based Custom Computing Machines*, pp. 22–28.

Walke R, Dudley J and Sadler D (2000) An fpga based digital radar receiver for soft radar *Proc. 34th Asilomar Conf. on Signals, Systems, and Computers*, pp. 73–78.

Kapasi U, Dally W, Rixner S, J.D. O and Khailany B (2002) Virtex5.pdf *Proc. 2002 IEEE Int. Conf. on Computer Design: VLSI in Computers and Processors*, pp. 282–288.

Kung HT and Leiserson CE (1979) Systolic arrays (for vlsi) *Sparse Matrix Proc. 1978*, pp. 256–282.

Kung SY (1988) *VLSI Array Processors*. Prentice Hall, Englewood Cliffs, NJ.

Laufer R, Taylor R and Schmit H (1999) PCI-Piperench and the SWORDAPI – a system for stream-based reconfigurable computing. *Proc. IEEE Symp. on Field-programmable Custom Computing Machines*, Napa, USA, pp. 200–208.

Li Z and Hauck S (2001) Configuration compression for virtex fpgas *Proc. IEEE Conf. on FPGA-based Custom Computing Machines*, pp. 147–159.

Luk W, Shirazi N, Guo SR and Cheung PYK (1997) Pipeline morphing and virtual pipelines. *7th Int. Workshop on Field-programmable Logic and Applications*, pp. 111–120.

Lysaght P and Stockwood J (1996) A simulation tool for dynamically reconfigurable field programmable gate arrays. *IEEE Trans. on VLSI Systems* 42(2), 381–390.

MacBeth J and Lysaght P (2001) Dynamically reconfigurable cores *Proc. IEEE Conf. on FPGA-based Custom Computing Machines*, pp. 462–472.

Maxfield C (2004) *The Design Warrior's Guide to FPGAs*. Newnes, Burlington.

Scalera SM and Vazquez JR (1998) The design and implementation of a context switching fpga *Proc. IEEE Conf. on FPGA-based Custom Computing Machines*, pp. 78–85.

Semiconductor Industry Association (2005) International technology roadmap for semiconductors: Design. Web publication downloadable from http://www.itrs.net/Links/2005ITRS/Design2005.pdf.

Todman T, Constantinides G, Wilton S, Cheung P, Luk W and Mencer O (2005) Reconfigurable computing: architectures and design methods. 152(2), 193–205.

Trimberger S, Carberry D, A. J and Wong J (1997) A time-multiplexed fpga *Proc. IEEE Conf. on FPGA-based Custom Computing Machines*, pp. 22–28.

Walke R, Dudley J and Sadler D (2000) An fpga based digital radar receiver for soft radar *Proc. 34th Asilomar Conf. on Signals, Systems, and Computers*, pp. 73–78.